Einführung in die Informationstechnik für Ingenieure

– Vorlesungsskript –

Herrmann–Föttinger–Institut für Strömungsmechanik
Technische Universität Berlin

W. Baumann, C. Böhning
T. Schmidt, L. Tornow

17. April 2003

Inhaltsverzeichnis

1 Allgemeine Einführung **1**

1.1 Mikroelektronik 1

1.2 Anwendung der Mikroelektronik 4

1.3 Zusammenfassung 9

2 Programmiertechnik **10**

2.1 Was ist ein Programm? 10

2.2 Was ist Programmieren? 11

2.3 Beispiel einer Problemanalyse 13

2.4 Anforderungsdefinition (Pflichtenheft) 14

2.5 Programmentwurf 16

2.6 Anweisungsentwurf mit Struktogrammen 17

2.6.1 Grundelemente 17

2.6.2 Fallunterscheidung 18

2.6.3 Schleifen 19

2.6.4 Verfeinerung 21

2.7 Test, Dokumentation, Wartung 21

2.7.1 Test 21

2.7.2 Dokumentation 22

2.7.3 Wartung und Pflege 22

2.8 Programmierstil – wie schreibt man ein gutes Programm? 22

2.9 Zusammenfassung 24

2.10 Fragen und Übungen zu Kapitel 2 25

2.11 Antworten zu den Fragen und Übungen zu Kapitel 2 26

3 Betriebssysteme **27**

3.1 Aufgabe, Struktur und Dienste eines Betriebssystems 27

3.1.1 Der Betriebssystemkern — Kernel 28

3.1.2 Verarbeitungsprogramme 29

3.1.3 Betriebssysteme für Mikrorechner 30

3.2 Betriebssystem UNIX 30

3.2.1 Geschichte und Philosophie von Unix 30

3.2.2 Rechnerzugang 32

3.2.3 Das Dateisystem 33

3.2.3.1 Dateiarten 33

3.2.3.2 Dateibaum 34

3.2.3.3 Umgang mit Dateien 36

3.2.3.4 Eigentümer, Gruppen und andere Nutzer 38

3.2.4 Die Prozessverwaltung 41

3.2.5 Das Speichermanagement 44

3.2.6 Die Benutzerschnittstelle — die Shell 45

3.2.6.1 Komfortable Kommandoeingabe 46

3.2.6.2 Der History–Mechanismus der C–Shell 46

3.2.6.3 Dateinamenexpansion bei Verwendung von Metazeichen 47

3.2.6.4 Umlenkungsmechanismen 48

3.2.6.5 Filter 48

3.2.6.6 Pipes 49

3.2.6.7 Drucken 49

3.2.6.8 Online–Dokumentation 50

3.2.7 Vernetzung 52

3.2.7.1 NFS — NIS 52

3.2.7.2 Einloggen auf entfernten Rechnern 52

3.2.7.3 World Wide Web — WWW 53

3.2.7.4 File Transfer Protocol — FTP 54

3.2.7.5 Elektronische Post — E–Mail 55

3.2.8 X–Window — eine grafische Benutzeroberfläche 55

3.2.8.1 Einführung 55

3.2.8.2 Server, Client und Kommunikationskanal . . . 56
3.2.8.3 Windowmanager . . . 57
3.2.8.4 Benutzung von X . . . 58
3.2.8.5 Weitere Anwendungsprogramme unter X . . . 59
3.2.9 Editor — vi . . . 59
3.2.9.1 Aufruf und Verlassen des Editors . . . 59
3.2.9.2 Cursorpositionierung . . . 60
3.2.9.3 Einfügen, Verändern und Löschen . . . 62
3.2.9.4 Suchen und Ersetzen, reguläre Ausdrücke . . . 63
3.3 Programmübersetzung . . . 64
3.3.1 Compiler . . . 64
3.3.2 Linker und Lader . . . 64
3.3.3 Die Compiler f95 / gcc . . . 66
3.3.4 Werkzeuge zur Programmerstellung — make . . . 66
3.3.4.1 Makefile . . . 67
3.4 Zusammenfassung . . . 68
3.5 Fragen und Übungen zu Kapitel 3 . . . 69
3.6 Antworten zu den Fragen und Übungen zu Kapitel 3 . . . 70

5 Darstellung von Zahlen und Zeichen im Rechner 72

5.1 Darstellung von Zeichen . . . 72
5.1.1 ASCII–Kodierung . . . 72
5.2 Zahlensysteme . . . 73
5.2.1 Dezimalzahlen . . . 74
5.2.2 Dualzahlen / Binärzahlen . . . 74
5.2.3 Hexadezimalzahlen . . . 74
5.3 Darstellung von Zahlen im Rechner . . . 75
5.3.1 Natürliche Zahlen . . . 75
5.3.2 Ganze Zahlen — INTEGER . . . 75
5.3.2.1 Darstellung mit Vorzeichenbit . . . 75
5.3.2.2 Darstellung im Zweierkomplement . . . 75
5.3.2.3 127–Exzeßdarstellung . . . 76
5.3.3 Reelle Zahlen — REAL . . . 76

5.3.3.1 REAL–Zahlen nach Mikroprozessornorm . . . 76
5.3.3.2 Erhöht genaue Zahlendarstellung . . . 77
5.3.3.3 Genauigkeit der internen Zahlenrepräsentation . . . 78
5.3.4 BCD–Kodierung dezimaler Zahlen . . . 78
5.4 Zusammenfassung . . . 79
5.5 Fragen und Übungen zu Kapitel 5 . . . 80
5.6 Antworten zu den Fragen und Übungen zu Kapitel 5 . . . 80

6 Rechner – Aufbau **81**
6.1 Aufgaben eines Rechners . . . 81
6.2 Rechner–Arbeitsplatz . . . 82
6.3 Motherboard . . . 83
6.3.1 Der Prozessor . . . 85
6.3.2 Der Arbeitspeicher . . . 87
6.3.3 Der DMA–Kanal . . . 87
6.3.4 Die Tastaturschnittstelle . . . 88
6.3.5 Der Prozessorbus . . . 88
6.3.6 Die Interrupts . . . 88
6.4 Peripherie . . . 89
6.4.1 Plattenlaufwerke . . . 89
6.4.2 Grafikkarte und Bildschirm . . . 91
6.4.3 Ausgabegeräte und Kommunikation . . . 92
6.4.4 Eingabegeräte . . . 94
6.4.5 Speichermedien . . . 95
6.4.6 Zusätzliche Hardware . . . 95
6.4.6.1 Koprozessoren . . . 96
6.4.6.2 Soundkarten . . . 96
6.4.6.3 Videokarten . . . 97
6.4.6.4 Analog–Digital–Wandler . . . 97
6.4.6.5 Digital–Analog–Wandler . . . 98
6.4.6.6 Digitale Signalprozessoren (DSPs) . . . 98
6.5 Zusammenfassung . . . 99
6.6 Fragen und Übungen zu Kapitel 6 . . . 100
6.7 Antworten zu den Fragen und Übungen zu Kapitel 6 . . . 101

7 Mikroprozessoren 102

7.1 Allgemeiner Aufbau und Funktion . . . 102

7.1.1 Struktur . . . 102

7.1.1.1 Register . . . 103

7.1.1.2 Stack . . . 104

7.1.1.3 Steuerung . . . 104

7.1.2 Von–Neumann–Rechner . . . 105

7.1.3 Operationen . . . 107

7.1.3.1 Speicherschreib- und Leseoperationen . . . 107

7.1.3.2 Ein-/Ausgabeoperationen . . . 107

7.1.4 Interruptssystem . . . 108

7.1.5 Bussteuerung . . . 108

7.1.6 Adressierung . . . 109

7.2 Befehlssatz . . . 110

7.2.1 Datentransportbefehle . . . 110

7.2.2 Blocktransfer- und Suchbefehle . . . 111

7.2.3 Arithmetikbefehle . . . 111

7.2.4 Logikbefehle . . . 111

7.2.5 Schiebe- und Rotationsbefehle . . . 112

7.2.6 Bitverarbeitende Befehle . . . 112

7.2.7 Ein- und Ausgabefehle . . . 112

7.3 Steuerung . . . 113

7.3.1 Programmsteuerung . . . 113

7.3.2 Prozessorsteuerung . . . 113

7.3.3 Systemsteuerung . . . 113

7.4 Der 8–Bit–Mikroprozessor Intel 8080 . . . 113

7.5 Architekturerweiterung und moderne Konzepte . . . 115

7.5.1 Erhöhung der Wortbreite . . . 116

7.5.2 Vergrößerung des Adreßraums . . . 116

7.5.3 Virtuelle Speichertechnik . . . 117

7.5.4 Zeitlich verschränkte Befehlsausführung . . . 117

7.5.5 Cache . . . 118

7.5.6 Parallelverarbeitung . . . 118
7.5.7 RISC–Architektur . . . 119
7.5.8 Materialien . . . 121
7.6 Zusammenfassung . . . 121
7.7 Fragen und Übungen zu Kapitel 7 . . . 123
7.8 Antworten zu den Fragen und Übungen zu Kapitel 7 . . . 124

8 Rechnerschnittstellen **126**
8.1 Übertragung von Daten . . . 126
8.1.1 Betriebsarten . . . 127
8.1.2 Serielle und parallele Übertragung . . . 127
8.2 Die Centronics–Schnittstelle . . . 128
8.3 Die V.24–Schnittstelle . . . 130
8.4 Weitere Schnittstellen: Der IEC–Bus . . . 133
8.4.1 Funktionsweise des IEC–Busses . . . 134
8.4.2 Funktionale Eigenschaften . . . 134
8.4.3 Handshakeverfahren . . . 135
8.5 Zusammenfassung . . . 135
8.6 Fragen und Übungen zu Kapitel 8 . . . 137
8.7 Antworten zu den Fragen und Übungen zu Kapitel 8 . . . 137

9 Netzwerke **138**
9.1 Allgemeine Einführung . . . 138
9.2 Die Architektur von Computer–Netzwerken . . . 139
9.3 Klassifizierung von Rechnernetzen . . . 140
9.3.1 Nach der Topologie . . . 141
9.3.2 Nach der Vermittlungstechnik . . . 141
9.3.3 Nach der Reichweite . . . 141
9.3.3.1 Lokale Netze — Local Area Networks (LAN) . . . 142
9.3.3.2 Fernnetze — Wide Area Networks (WAN) . . . 143
9.4 Das OSI–Schichtenmodell . . . 143
9.4.1 Schicht 1 — Bitübertragungsschicht . . . 144
9.4.2 Schicht 2 — Abschnittssicherungsschicht . . . 144

9.4.3 Schicht 3 — Vermittlungsschicht . . . 144
9.4.4 Schicht 4 — Transportschicht . . . 144
9.4.5 Schicht 5 — Kommunikationssteuerungsschicht . . . 145
9.4.6 Schicht 6 — Darstellungsschicht . . . 145
9.4.7 Schicht 7 — Anwendungsschicht . . . 145
9.5 Das Internet–Schichtenmodell . . . 145
9.6 Netzwerkstandards . . . 146
9.6.1 Die IEEE Standards für lokale Netze . . . 147
9.6.2 Beispiel einer Anwendung der Schicht 3: IP . . . 147
9.6.3 Beispiel einer Anwendung der Schicht 4: TCP . . . 148
9.7 Zusammenfassung . . . 148
9.8 Fragen und Übungen zu Kapitel 9 . . . 149
9.9 Antworten zu den Fragen und Übungen zu Kapitel 9 . . . 149

10 Grafische Datenverarbeitung 150

10.1 Allgemeine Einführung . . . 150
10.2 Die grafische Darstellung . . . 151
10.3 Das Grafiksystem . . . 151
10.4 Standards der Computergrafik . . . 152
10.4.1 PHIGS und OpenGL . . . 152
10.4.2 PGPLOT . . . 154
10.5 PostScript . . . 156
10.5.1 Portable Document Format (PDF) . . . 158
10.5.1.1 Überblick . . . 158
10.5.1.2 Acrobat . . . 158
10.5.1.3 Anwendungsbereiche . . . 158
10.6 Zusammenfassung . . . 159
10.7 Fragen und Übungen zu Kapitel 10 . . . 159
10.8 Antworten zu den Fragen und Übungen zu Kapitel 10 . . . 160

11 Textverarbeitung 161

11.1 Allgemeine Einführung . . . 161
11.2 Formen der Textverarbeitung . . . 162

11.2.1 Der Editor 162
11.2.2 Das Textverarbeitungsprogramm 162
11.2.3 Der Textformatierer 162
11.2.4 Das Desktop–Publishing–Werkzeug 162
11.3 Auswahl einer Textverarbeitung 163
11.4 Textverarbeitung unter UNIX 164
11.4.1 LaTeX 164
11.4.1.1 Was ist LaTeX ? 164
11.4.1.2 Arbeitsweise 165
11.4.1.3 Steueranweisungen 165
11.4.1.3.1 Einige wichtige Steueranweisungen: 165
11.4.1.4 Verschiedene Schrifttypen 166
11.4.1.5 Erstellen von Tabellen 166
11.4.1.6 Mathematischer Modus 167
11.4.1.6.1 Die Begrenzer 167
11.4.1.6.2 Griechische Buchstaben 167
11.4.1.6.3 Exponenten und Indizes 167
11.4.1.6.4 Wurzeln 167
11.4.1.6.5 Mathematische Operatoren 168
11.4.1.7 Beispiel 168
11.4.2 Erstellen eines LaTeX–Dokumentes 169
11.5 Zusammenfassung 169
11.6 Fragen und Übungen zu Kapitel 11 170
11.7 Antworten zu den Fragen und Übungen zu Kapitel 11 170

Literatur **173**

Kapitel 1

Allgemeine Einführung

Dieses Kapitel soll anhand einer kurzgefassten Darstellung der historischen Entwicklung und einiger Definitionen einen allgemeinen Einstieg in die Informationstechnik vermitteln.

Informationstechnik kann man als angewandte Informatik bezeichnen. Eine genauere Beschreibung folgt später.

Zentrale Rollen in der Informationstechnik spielen die Geräte („Hardware“) und damit die Mikroelektronik einerseits und die Programme („Software“) andererseits.

1.1 Mikroelektronik

Die Mikroelektronik befasst sich mit der Entwicklung, Produktion und Anwendung mikroelektronischer Bauelemente und Schaltungsgruppen. Dabei ist auch die Grundlagenforschung zur

- Gewinnung hochreiner Materialien
- Realisierung vielfältiger technischer Verfahren (z.B. Ätzverfahren)

wichtig.

Dominierende Bauelemente sind mikroelektronische Schaltkreise (ICs, „Integrated Circuits“). Sie enthalten auf einer Fläche von $1mm^2$ bis $1cm^2$ eine große Anzahl von Transistorfunktionen (moderne Mikroprozessoren wie der Pentium4 von Intel $\approx$ 55 Mio.) mit Strukturelementen im μm Bereich (modernste Speicher wie 256–MBit–Chips im Sub–μm–Bereich). Die Integration zu komplexen Funktionsgruppen geschieht durch gemeinsame Herstellung vieler Transistoren auf einer Siliziumscheibe („Wafer“) und deren anschließende „Verdrahtung“ durch aufgedampfte Substanzen zu Schaltungen (Logikgatter, Speicher, Mikroprozessoren, etc.).

Anfang der siebziger Jahre begann die Entwicklung des Mikroprozessors, d.h. programmierbarer Elektronik. Diese ist besonders interessant, da Mikroprozessoren universell einsetzbare Schaltkreise sind und daher in großen Stückzahlen produziert und verwendet werden. Die Anpassung an spezielle Aufgaben geschieht über eine programmierte Befehlsfolge. Die Forderung der Rechnerhersteller nach billigen Logikschaltkreisen und Halbleiterspeichern übte starken Einfluss auf die Mikroelektronik–Technologie aus. Mikroprozessoren und auf ihrer Grundlage gefertigte Mikrorechner und Steuerungseinheiten ermöglichen den Einsatz programmgesteuerter Geräte auch dort, wo vorher z.B. aus Kostengründen der Einsatz eines Kleinrechners oder eines aus diskreten Logikelementen aufgebauten Systems für Steuerungs-, Mess- oder Automatisierungszwecke nicht in Frage kam.

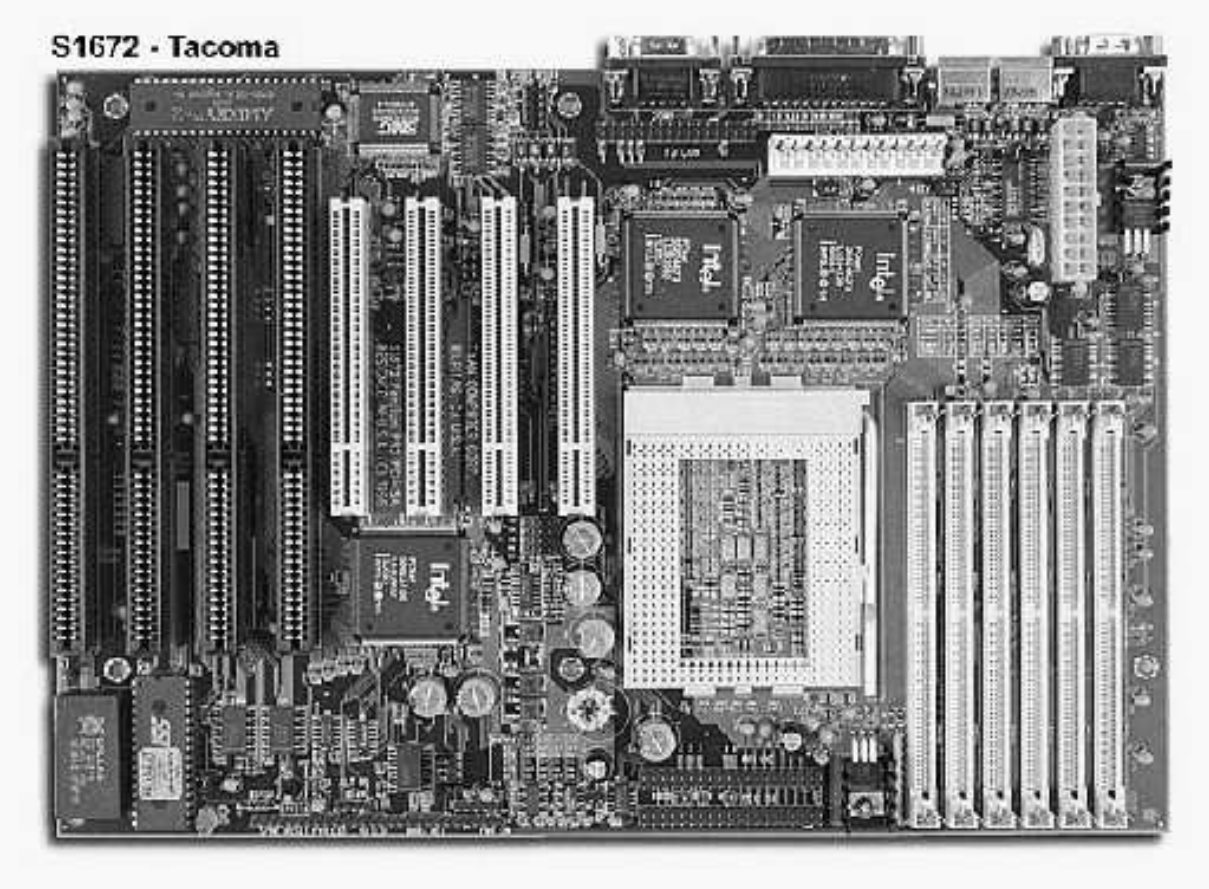


Abbildung 1.1: Das S1672–Motherboard der Firma Tacoma [18]

Seit einigen Jahren geht ein neuerer Trend zu anwenderspezifischen Schaltkreisen (Application Specific ICs„ASICs“). Sie werden für spezielle Aufgaben verwendet und stellen eine Kombination von Standard–ICs dar. So war zum Beispiel die Grundplatine („Motherboard“) eines Original–IBM–AT vor wenigen Jahren mit etwa 50 Chips (Speicherbausteine ausgenommen) bestückt. Heute kommt man mit Sätzen von 3 oder 4 ASICs plus einige wenige zusätzliche Chips aus. Abbildung 1.1 zeigt ein Motherboard mit 3 Chips der Firma Intel, die die wesentlichen Steuerungsaufgaben für das Zusammenspiel der einzelnen Komponenten ausführen. Weitere Bestandteile der Platine sind: in der Mitte der Platz (ZIF Socket 8) für den Prozessor, rechts davon 6 Steckplätze für Speichermodule (72pin–SIMM–Sockets), links vom Prozessorsockel sind 4 PCI– und 4 ISA–Bus–Steckplätze für Erweiterungskarten. Dazu gehören z.B. Grafikkarten, Soundkarten, Ethernetkarten u.a.

Etwa alle 1,5 bis 2 Jahre verdoppelt sich der Integrationsgrad (Anzahl der auf einem Chip realisierten Transistorfunktionen; Moore's Law (Gordon Moore, 1965) geht von einer Verdoppelung der Transistoren alle 18 Monate aus). Dies wird verursacht durch Reduktion der Strukturabmessungen, Nutzung neuer Effekte und Schaltungsprinzipien, sowie auch durch größere Chipflächen (bis $\sim 1cm^2$). Der Alpha 21164 der Digital Equipment Corp. hat 9,6 Millionen Transistoren und erreicht damit bei einer Taktrate von 600MHz eine Rechenleistung von 2,4 BIPS — **b**illions (Milliarden) of **i**nstructions **p**er **s**econd). Gleichzeitig fallen die Kosten für die Rechenleistung.

Die Mikroelektronik zeigt ein beispiellos hohes Wachstum über ca. 25 Jahre ohne erkennbare Sättigungseffekte (Es gibt dennoch physikalische Grenzen, wenn z.B. elektromagnetische Effekte von quantenmechanischen Effekten übertroffen werden). Der Fortschritt in der Halbleitertechnologie ist beispielhaft in

Das „Herz“ eines Rechners bildet die CPU und insbesondere durch die stetige und schnelle Entwicklung auf diesem Gebiet, sind die heutigen Leistungen von Computern möglich. Die Abbildung 1.3 zeigt einen modernen Prozessor der Intel Corp., der auf einer sehr geringen Fläche mehrere Millionen Transistoren vereint.

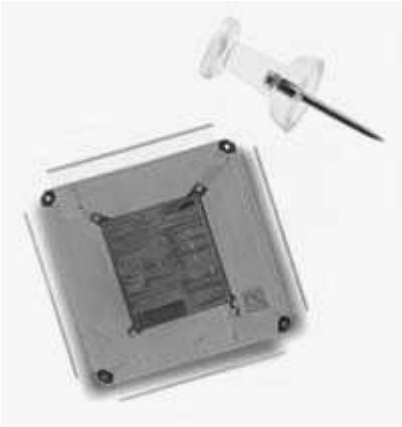


Abbildung 1.3: Größe eines Prozessors [20]

Tabelle 1.1 aufgeführt.

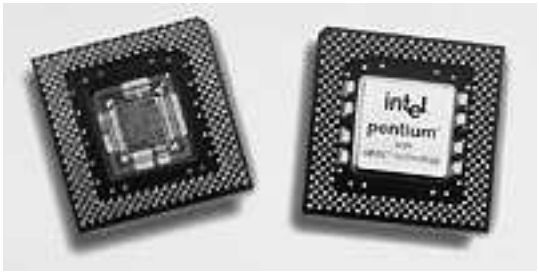


Abbildung 1.2: Prozessor [19]

Nicht nur die Anzahl der Transistoren erhöhte sich im Verlauf der vergangenen 25 Jahre, sondern auch Rechengeschwindigkeiten. Es gibt mehrere Größen, die Leistungsfähigkeit eines Mikroprozessors zu messen. Eine gebräuchliche ist die mögliche Anzahl der ausgeführten Instruktionen pro Sekunde — MIPS (engl. **m***illions of* **i***nstructions* **p***er* **s***econd*)

Jahr	Chip	MIPS	Taktrate MHz	Transistorzahl
1971	4004	0,06	0,108	2.300
1972	8008	0,06	0,200	3.500
1974	8080	0,64	2,000	6.000
1976	8085	0,37	5,000	6.500
1978	8086	0,75	10,000	29.000
1982	80286	2,66	12,000	134.000
1985	386DX	11,40	33,000	275.000
1989	486DX	41,00	50,000	1.200.000
1992	486DX2	54,00	66,000	1.200.000
1993	Pentium66	112,00	66,000	3.100.000
1994	Pentium100	166,30	100,000	3.200.000
1995	Pentium133	218,90	133,000	3.300.000
1996	Pentium166		166,000	3.300.000
1997	Pentium233		233,000	4.500.000
1997	PentiumPro	440,00	200,000	5.500.000
1997	PentiumII		300,000	7.500.000
1999	PentiumIII		600,000	9.500.000
2000	Pentium4		1.500,000	42.000.000
2001	Pentium4		1.800,000	42.000.000
2002	Pentium4		2.200,000	55.000.000

Tabelle 1.1: Entwicklung von Prozessoren der Intel Corp. (Auswahl)
Quelle des Zahlenmaterials: http://www.intel.com/

1.2 Anwendung der Mikroelektronik

Die Mikroelektronik hat mittlerweile in allen Bereichen des gesellschaftlichen Lebens Einzug gehalten. Sie wird angewendet in:

- Informationsgewinnung, -übertragung, -verarbeitung, und -speicherung
- Steuerung, Optimierung und Automatisierung von Produktions- und Transportprozessen aller Art
- Nachrichtentechnik
- Simulation in Industrie und Forschung
- Grundlagen– und Angewandte Forschung
- Heimelektronik
- Umweltschutz
- Verkehrstechnik

Der Kern eines Steuerungs- oder Verarbeitungssystems wird dabei oft von einem Mikroprozessor gebildet. Der Mikroprozessor hat tiefgreifende Veränderungen in der Arbeitswelt bewirkt, s. [3]:

- den Ersatz der Registrierkasse durch elektronische Kassensysteme, die automatisch Preisschilder abtasten, den Verkauf einer Ware im Lagerbestand verbuchen und die Buchführung unterstützen.
- die Automatisierung der Arbeit von Setzern; das Zusammensetzen einer Zeitungsseite Buchstabe für Buchstabe entfällt, das Setzen kann direkt vom Journalisten durch Eingabe der Texte in den Computer vorgenommen werden.

- den Ersatz der Schreibmaschine durch Textverarbeitungssysteme, auf denen Texte gestaltet, standardisiert und beliebig korrigiert werden können.

Das umfangreichste *Einsatzfeld* der Mikroelektronik ist die *Informatik*. Sie ist eine Wissenschaftsdisziplin mit Querschnittscharakter und untersucht Gesetzmäßigkeiten bei

- Gewinnung
- Bearbeitung
- Speicherung
- Übertragung und
- Anwendung von Informationen in Produktion, wissenschaftlicher, gesellschaftspolitischer und kultureller Tätigkeit des Menschen.

Die Informatik umfasst somit den Wissenschaftsbereich (Computerwissenschaft, engl. *computer science*) und die Anwendung (EDV=Elektronische Datenverarbeitung, engl. *data processing*). Im Laufe der Entwicklung der Rechentechnik wird der Informatik–Begriff zusammenfassend für die speziellen Gebiete der elektronischen Informationsverarbeitung gebraucht. Informatik umfasst „Fragen des Entwurfs, der Konstruktion, der Entwicklung sowie der Anwendung und der Instandhaltung von Informationsverarbeitungssystemen (Rechner, DV–Anlagen, PC, etc.), Speicher- und Kommunikationssystemen einschließlich Hard- und Software wie auch organisatorische Aspekte und Probleme der Mensch–Maschine–Kommunikation. ... Die Informatik schafft die Grundlagen für moderne Informations- und Kommunikationstechnologien sowie die methodischen und theoretischen Mittel zur Anwendung der Mikroelektronik in Informationsübertragungssystemen“ [10].

Es ist klar, dass die Entwicklung und Anwendung der Informatik eng verknüpft ist mit der Verfügbarkeit von Rechnern und Rechnerperipherie unterschiedlichster Leistungsklassen.

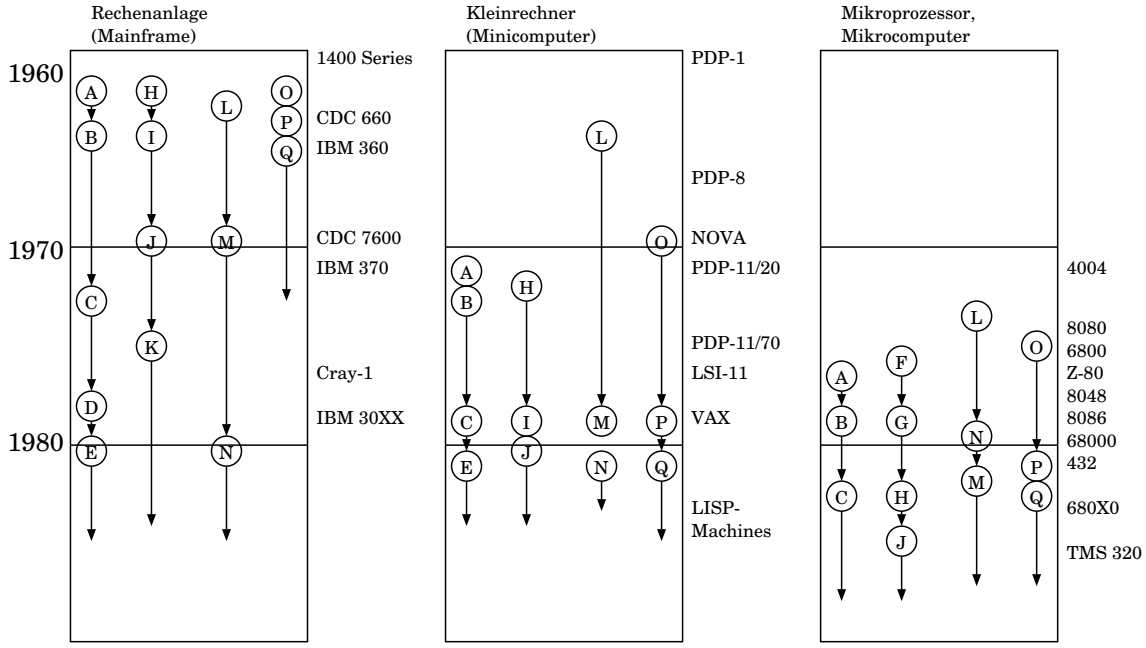
Die Anforderungen der Informatik stimulieren die Entwicklung immer komplexerer ICs und die Suche nach effektiven Rechnerarchitekturen. Auch die Entwicklung von Rechnerperipherie ((Massen-)Speicher, Ausgabegeräte, Datenkommunikation, etc.) ist davon stark beeinflusst. Dadurch bekommt der Anwender (Ingenieur, Natur-, Sozial-, Wirtschafts-, Geisteswissenschaftler, Kaufmann, Verwaltungsflachmann, etc.) mit Personal–Computern und Arbeitsplatzrechnern (engl. *workstation*) leistungsfähige Mittel zur intensiven Nutzung der Informatik und Rechentechnik für seine eigenen Aufgaben.

Die schnelle Entwicklung der Informatik in den letzten Jahren basiert andererseits auch auf dem hohen Wachstumstempo der Mikroelektronik. Die Entwicklung beider Bereiche — der Informatik und der Mikroelektronik — beeinflussen sich gegenseitig.

Das Wachstum resultiert aus leistungsfähigen rechnergestützten Entwurfsverfahren und aus der Realisierung von Architekturmerkmalen, die zuvor nur bei leistungsfähigen Mini- oder Großrechnern zu finden waren.

Parallel dazu ist die Entwicklung auf dem Gebiet der Software zu sehen, die durch den rasanten Fortschritt der Mikroelektronik stimuliert wurde. So wächst der Leistungsumfang, die Komplexität und die Bedienerfreundlichkeit verfügbarer Software (höhere Programmiersprachen, Betriebssysteme, Anwendungs-Software) ständig.

Der Mikrorechnerbereich profitiert natürlich auch von den Entwicklungen im Bereich größerer Rechner. Die wesentlichen Stufen in der Entwicklung von Hard- und Software für Mainframes und Minicomputer werden auch bei Mikrorechnern durchlaufen, dank schneller Entwicklung der Mikroelektronik aber in erheblich kürzerer Zeit, siehe Abb. 1.4.



Einführung von Neuerungen in der Architektur und Software von Rechenanlagen.

Software:	A:	höhere Programmiersprachen allgemein üblich,
	B:	Segmentierung, Mehrnutzersysteme
	C:	virtuelle Speichertechnik,
	D:	virtuelle Dateisysteme,
	E:	Unterstützung von nichtprozeduralen Sprachen.
Arithmethik:	F:	16–Bit–Integerzahlen,
	G:	32–Bit–Integerzahlen,
	H:	Gleitkommaformat,
	I:	interne Datenkonversion,
	J:	Pipeline–Verarbeitung, Array–Prozessoren u.ä.,
	K:	Vektor/Matrix–Arithmetik.
Anwendung:	L:	variable Konfiguration,
	M:	modifizierbarer Mikrocode, an den Anwendungsfall angepasste Maschinenbefehle,
	N:	Verlagerung von Software in die Hardware.
Ein-/Ausgabe–Strukturen:	O:	direkter Speicherzugriff DMA, speicherbezogene Ein-/Ausgabe–Adressierung,
	P:	Datenkanäle, „intelligente“ Ein-/Ausgabe,
	Q:	Multiprozessoren.

Abbildung 1.4: Entwicklungsetappen für Großrechner (Mainframe), Kleinrechner(Minicomputer) und Mikrorechner/ Mikroprozessoren [10]

Die folgende Tabelle gibt einen kleinen historischen Überblick zur Entwicklung der Rechentechnik und Mikroelektronik, insbesondere im Bereich der Personal Computer.

Ausgewähltes zur Entwicklung der Rechentechnik und Mikroelektronik ([10],[1, 9/90], WWW)

1623	Wilhelm Schickard (1592–165): funktionierende Rechenmaschine mit Zahnradgetriebe für die vier Grundrechenarten und mit automatischem Zehnerübertrag
1642	Blaise Pascal (1623–1662): Rechenmaschine für achtstellige Additionen und Subtraktionen
1673	Gottfried Wilhelm von Leibniz (1646–1716): mechanische Rechenmaschine mit Staffelwalze für die vier Grundrechenarten; erst 1894 funktionstüchtiges Exemplar
1679	duales Zahlensystem und Gesetze der binären Arithmetik
1727	Antonius Braun (1685–1728): funktionstüchtige Sprossenradmaschine für die vier Grundrechenarten
1774	Philipp Matthäus Hahn (1739–1790): funktionstüchtige Rechenmaschine nach dem Prinzip der Staffelwalze für 4 Grundrechenarten, 11 Stellen
1808	Joseph-Maria Jacquard (1752–1834): Webstuhlsteuerung mit Lochkarten
1833	Charles Babbage (1792–1871): erstes Konzept eines programmgesteuerten Rechenautomaten mit den Baugruppen: • Rechenwerk für 4 Grundrechenarten (dekadisches Zählrad), • Zahlenspeicher, • Lochkartensteuereinheit, • Dateneingabegerät, • Datenausgabegerät mit Druckwerk Mit den technischen Mitteln in der Mitte des 19. Jahrhunderts nicht realisierbar; späterer Nachbau bestätigte die Funktionstüchtigkeit der Konstruktion von Babbage
1855	George Boole (1815–1884): Begründung der zweiwertigen Algebra für Aussagen mit den Werten 0 und 1
1874	Karl Ferdinand Braun (1850–1918): Entdeckung des Halbleitereffekts (Kristalldetektor)
1906	Lee de Forest (1873–1961), Robert von Lieben (1878–1913): Erfindung der Elektronenröhre (Triode)
1938	Claude Elwood Shannon (geb. 1916): Formulierung der Schaltalgebra 1948: Wesentliche Beiträge zur Begründung der Informationstheorie
1941	Konrad Zuse (1910–1995): Vorführung des ersten programmgesteuerten Rechenautomaten ZUSE Z3 (Dualsystem, Relaistechnik); Konzept einer Programmiersprache 1937: Versuchsanlage Z1 mit mechanischen Schaltgliedern
1942	John V. Atanasoff (1903–1995): Rechenanlage mit Elektronenröhren
1944	Howard H. Aiken (1900–1973): in den USA Inbetriebnahme des elektromechanischen programmgesteuerten Rechenautomaten MARK I
1945	John von Neumann (1903–1957): Konzeption eines speicherprogrammierten Rechenautomaten (EDVAC)
1945/47	John Presper Eckert (1919–1995), John William Mauchly (1907–1980): vollelektronische Rechenanlage ENIAC
1947	John Bardeen, Walter H. Brattain, William Shockley: Erfindung des Transistors
1955	Transistoren für Rechenschaltungen eingesetzt
1957	FORTRAN–Compiler
1958	J.S. Kilby: Erfindung der integrierten Schaltung
1961	Fairchild: erster monolithischer Schaltkreis
1965	Digital Equipment Corp.: Minirechner PDP–8
1965	Programmiersprache BASIC

1969	Betriebsystem UNIX
1970	Erster Taschenrechner IBM stellt mit der Serie 370 die ersten Rechner mit Mikroprogrammierung vor. Sie sind noch heute Standard
1971	Intel Corp.: Chip–Satz mit 4–Bit–Mikroprozessor 4004; Programmiersprachen PASCAL und C entwickelt
1971/72	Intel Corp.: erster 8–Bit–Mikroprozessor 8008
1973	Intel Corp.: 8–Bit–Mikroprozessor 8080; Scelbi Computer Consulting: erster Mikrocomputer auf der Basis des Intel 8008: Scelbi 8–H
1975	Entwicklung zahlreicher 8–Bit–Mikroprozessortypen (Zilog Z80, MOS Technology 6502, Motorola 6800) Erster „echter“ Mikrocomputer Altair 8800 mit Intel 8080
1976	Vorstellung des Mikrocomputers Apple I mit Mikroprozessor 6502; Erster 16–Bit–Mikroprozessor (Texas Instruments TMS9000)
1977	Vorstellung der Microcomputer Apple II und TRS–80 (Radio Shack); Betriebssystem CP/M für die erste Generation von Mikrocomputern von Digital Research entwickelt
1978	Mikrocomputer Atari 400 und Atari 800 vorgestellt; Epson stellt den MX–80 Nadeldrucker vor und revolutioniert damit den low–cost Drucker–Markt; Intel Corp.: 16–Bit–Mikroprozessor 8086
1979	VisiCalc, erstes Tabellenkalkulationsprogramm für Mikrocomputer vorgestellt; Intel Corp. stellt den 16–Bit–Mikroprozessor 8088 vor, der Herz des IBM PCs werden wird
1980	Vorstellung der Billig–Mikrorechner Sinclair ZX80 und Commodore VIC–20; Digital Research kündigt das Mikrocomputer–Betriebssystems CP/M–86 an
1981	IBM PC vorgestellt. Erste 32–Bit–Mikroprozessoren vorgestellt. Tragbarer Mikrocomputer Osborne 1 vorgestellt; Epson stellt den wahrscheinlich ersten Laptop–Computer, HX–20 vor; Hayes stellt das Smartmodem 300 vor, welches sich zum Industriestandard entwickeln wird
1982	Japan: Zielsetzung der 5. Rechnergeneration („künstliche Intelligenz“); Compaq Computer Corp. stellt den IBM–PC–kompatiblen Compaq Portable vor; Commodore stellt den Commodore 64 vor; Intel Corp. stellt den 16–Bit–Microprozessor 80286 vor; Columbia Data Products stellt den ersten IBM PC–Clone, MPC vor; Lotus Development stellt mit 1–2–3 sein Tabellenkalkulationsprogramm vor; Autodesk stellt AutoCAD, das erste CAD-Programm für PCs vor
1983	Einführung des IBM XT, einer Erweiterung des IBM PC um eine 10–MByte–Festplatte und mehrere Erweiterungs–Steckplätze; AT&T stellt das Betriebssystem Unix System V vor
1984	Motorola stellt den 32–Bit–Mikroprozessor 68020 vor; Apple stellt die Microcomputer Apple IIc und Macintosh vor; IBM stellt den AT mit dem 80286 Microprozessor vor
1985	Entwicklung von 1–MBit–DRAMs durch IBM sowie mehrere japanische Firmen; Atari stellt den 520ST vor; Commodore stellt den Amiga 1000 vor
1986	Vorstellung der ersten 80386 basierten PCs durch ALR und Compaq; Motorola stellt den 32–Bit–Microprozessor 68030 vor
1987	IBM entwickelt den ersten 4–MBit–DRAM-Chip; Compaq stellt den Portable III vor; Apple stellt den Mac SE und den Mac II vor; IBM bringt die PS/2–Reihe mit Microchannel–Architektur und das Betriebssystem OS/2 auf den Markt und stellt mit dem Modell 80 seinen ersten 80386 basierten Rechner vor
1988	Dell und Tandy stellen die ersten Microchannel–Clones vor; Maxtor stellt das erste wiederbeschreibbare magneto-optische Plattenlaufwerk vor; Der NeXT–Rechner von Steve Jobs wird vorgestellt; Robert T. Morris Jr. legt mit einem gutartigen „Wurm“–Virus etwa 6000 Rechner im amerikanischen ARPANET lahm
1989	Vorstellung der Unix–Workstations DECstation 3100 und Sun SPARCstation 1; Intel Corp. stellt den 80486 sowie den „Cray–on–a–chip“ i860 vor; Erster PC mit 80486 in England vorgestellt; ALR stellt den ersten 80486–EISA–Bus-Rechner vor
1990	Motorola stellt den Mikroprozessor 68040 vor; IBM stellt die RISC-Prozessor–basierte Unix–Workstation RS/6000 vor

1991 MIPS Technologies stellt den R4000 vor, ein 100MHz, 64–Bit RISC–Prozessor

1992 Intel Corp. stellt den 486DX2 mit 50MHz (41 MIPS) und 66MHz (54 MIPS) vor; Intel Corp führt den PCI–Bus–Standard ein (**P***eripheral* **C***omponent* **I***nterconnect*); Digital Equipment Corp. stellt den Alpha 21064 ein 64–Bit–Prozessor mit 150 MHz vor

1993 Digital Equipment Corp. hat den Alpha 21064 auf 200 MHz weiterentwickelt; Intel Corp. stellt den Pentium vor: 32–Bit–Register, 64–Bit–Datenbus, 3.1 Mio Transistoren, 66MHz → 112 MIPS

1994 Intel Corp. führt den Pentium mit 90MHz (149.8 MIPS) und 100MHz (166.3 MIPS) ein; MIPS Technologies stellt den R4400, einen 200MHz RISC–Prozessor vor; Digital Equipment entwickelt einen 300MHz Alpha–Chip mit ca. 1000 MIPS

1995 Intel Corp. stellt den 133MHz–Pentium mit 3.2 Mio Transistoren und 218.9 MIPS sowie den 200MHz Pentium Pro mit 5.5 Mio Transistoren vor; Digital Equipment zeigt einen 333MHz Alpha

1996 Digital Equipment Corp. entwickelt einen 400MHz und 500MHz Alpha–Prozessor (2.0 BIPS); Intel Corp. stellt die MMX–Technologie vor

1997 Intel Corp. stellt den PentiumII mit 300MHz und 7.5 Mio Transistoren vor; es werden die ersten Chips mit MMX–Erweiterungen geliefert; Digital Equipment Corp. zeigen den 21164 Alpha 9.6 Mio Transistoren → 2.4 BIPS

1999 Der PentiumIII - ein modernisierter PentiumII wird vorgestellt; Der Athlon Prozessor erscheint und zum ersten Mal hat AMD den schnellsten x86 Prozessor; Der Microsoft Prozess beginnt

2000 AMD's Prozessoren überschreiten als erste vor Intel die 1 GHz Grenze, der Pentium4 als Nachfolger der PentiumPro Architektur erscheint

Weitere und detailliertere Informationen können im World Wide Web (s. Kapitel 3) unter vielen verschiedenen Quellen gefunden werden. Da solche URL's nicht notwendigerweise immer gepflegt werden oder möglicherweise auch wieder verschwinden, werden keine Adressen angegeben. Stattdessen sei auf die bekannten WWW–Adressen gängiger Suchmaschinen verwiesen, auf denen man mit entsprechenden Stichwortsuchen sehr schnell zu den gewünschten Informationen auf aktuellem Stand gelangt. Mögliche Suchmaschinen (ohne Anspruch auf Vollständigkeit):

- www.yahoo.com, oder de.yahoo.com
- www.altavista.com
- www.unix-ag.uni-siegen.de/search
- www.fireball.de
- www.google.de

1.3 Zusammenfassung

Dieses Kapitel gibt einen allgemeinen Einstieg in die Informationstechnik. Es zeigt, dass die Entwicklung der Informationstechnik maßgeblich beeinflusst worden ist durch den rasanten Fortschritt auf dem Gebiet der Mikroelektronik.

Mikrorechner werden heute in nahezu allen Bereichen des täglichen Lebens eingesetzt. Sie sind sehr leistungsfähige und kostengünstige Kombinationen von Mikroelektronik (Hardware) und Programmanweisungen (Software), die die Systemfunktionen definieren.

Informationstechnik umfasst die speziellen Gebiete der elektronischen Informationsverarbeitung.

Kapitel 2

Programmiertechnik

2.1 Was ist ein Programm?

Ein Programm ist nach DIN 44300 definiert als eine zur Lösung einer Aufgabe vollständige Anweisung zusammen mit allen erforderlichen Vereinbarungen. Diese Definition sagt nichts darüber aus

- wer diese Anweisung als Prozessor auszuführen hat, ob Durchschnittsmensch, Spezialist, Organisation oder Rechnersystem;
- mit welchen Mitteln Anweisungen und Vereinbarungen beschrieben sind.

Die folgende Liste von Beispielen zeigt nicht nur sehr verschiedene Aufgaben, sondern auch sehr verschiedene Beschreibungsmittel:

- Schaltplan eines Fernsehers
- Landkarte mit eingezeichneter Fahrtroute
- Überweisung an eine Bank
- Formel zur Berechnung von Mittelwerten und Standardabweichung
- Anleitung zum Benutzen eines Telephones
- Noten für ein Orchester
- Rezept zum Backen eines Apfelstrudels.

An diesen Beispielen kann man sich auch klar machen, daß die in der Definition verlangte *Vollständigkeit* der Beschreibung sehr stark von den Fähigkeiten des Bearbeiters oder Prozessors abhängt. Wenn etwa Studierende das Summenzeichen $\sum$ noch nicht kennen, so wird die Formel für den Mittelwert um einiges umständlicher.

Da es unser Ziel ist, Programme für Rechnersysteme zu konstruieren, müssen wir deren Fähigkeiten kennen. Einerseits sind heutige Rechner so zuverlässig und leistungsfähig, andererseits sind die Aufgaben, die wir uns hier stellen, so bescheiden, daß wir folgenden Satz festhalten können:

> Jede Aufgabe, deren Vorgaben sich in angemessener Weise in Zahlen darstellen, also formalisieren lassen und deren Lösung in endlich vielen Schritten berechnet werden kann, ist mit einem Programm lösbar.

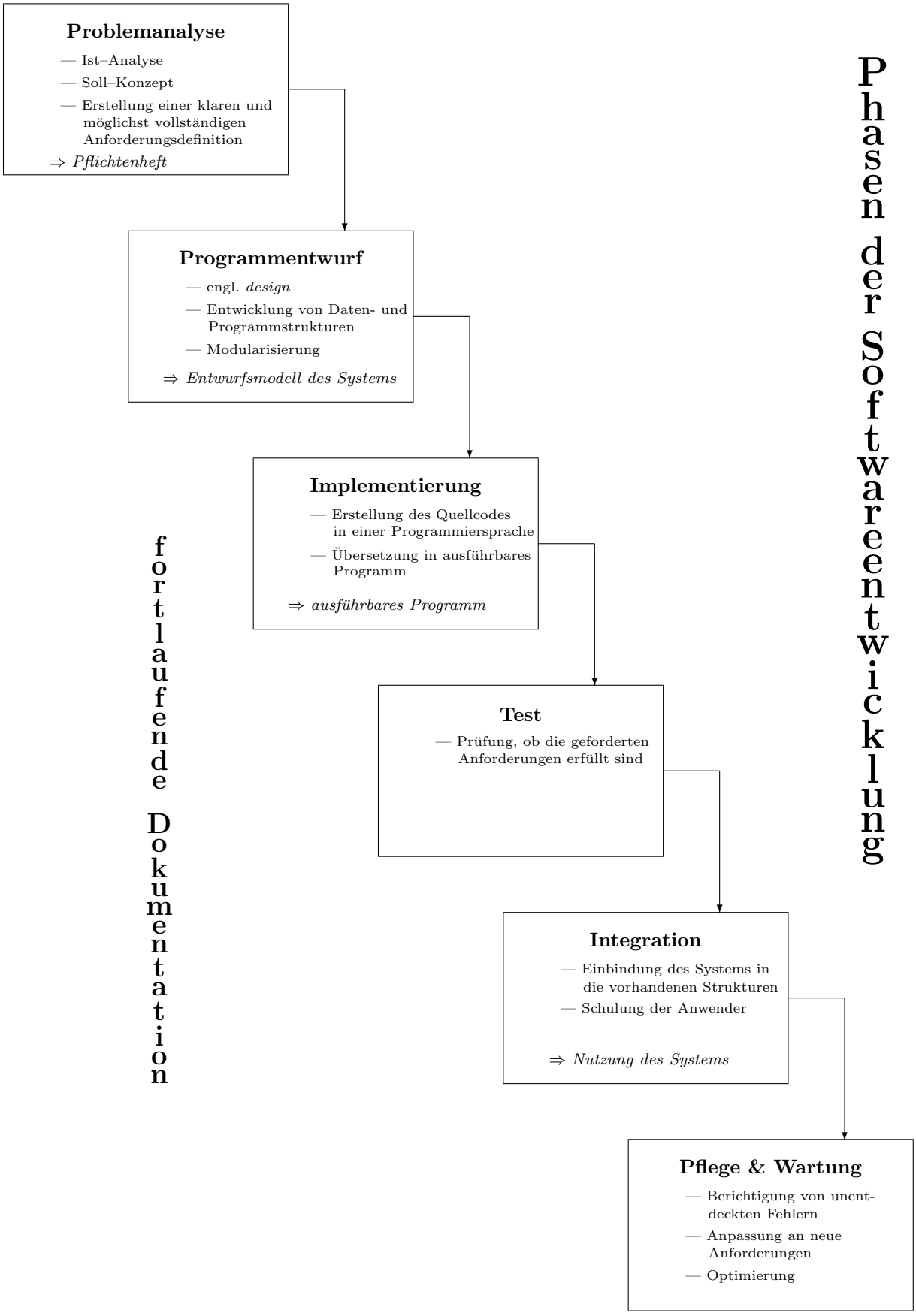
Dazu müssen wir wissen, wie wir dem Rechner Anweisungen zu geben haben, damit er für uns rechnet. Wesentlich schwieriger ist es zu wissen, wie sich Aufgaben in angemessener Weise formalisieren lassen.

2.2 Was ist Programmieren?

Aus dem vorigen Abschnitt ist deutlich geworden, daß beim Konstruieren eines Programms nicht die Umsetzung eines Lösungsentwurfs in Anweisungen an das Rechnersystem die schwierigste Arbeit ist, sondern das Entwerfen einer angemessenen Lösung für ein Problem. Es gibt verschiedene Vorgehensmodelle zur Entwicklung von Software. Wir wollen hier das Phasenmodell beschreiben. Es orientiert sich am Lebenszyklus von Software (engl. *software–lifecycle*). Die Phasen bauen grundsätzlich aufeinander auf, aber es gibt auch Rückkopplungen, die dazu führen können, daß Teile einer vorherigen Phase überarbeitet werden müssen.

Damit ein Softwaresystem von verschiedenen Personen bearbeitet werden kann und die Wartung mit vertretbarem Aufwand möglich ist, muß eine fortlaufende Dokumentation angefertigt werden. Das erleichtert auch die Fehlersuche und -berichtigung während der Erstellung eines Softwareproduktes.

Abschnitt 2.6 zeigt eine Methode zum Programmentwurf — die Nutzung von Struktogrammen.


Problemanalyse
— Ist–Analyse
— Soll–Konzept
— Erstellung einer klaren und möglichst vollständigen Anforderungsdefinition
⇒ Pflichtenheft
Programmentwurf
— engl. design
— Entwicklung von Daten- und Programmstrukturen
— Modularisierung
⇒ Entwurfsmodell des Systems
Implementierung
— Erstellung des Quellcodes in einer Programmiersprache
— Übersetzung in ausführbares Programm
⇒ ausführbares Programm
Test
— Prüfung, ob die geforderten Anforderungen erfüllt sind
Integration
— Einbindung des Systems in die vorhandenen Strukturen
— Schulung der Anwender
⇒ Nutzung des Systems
Pflege & Wartung
— Berichtigung von unentdeckten Fehlern
— Anpassung an neue Anforderungen
— Optimierung
Phasen der Softwareentwicklung
fortlaufende Dokumentation

Damit läßt sich Programmieren, also das Aufsuchen einer Problemlösung und ihre vollständige Beschreibung, gut vergleichen mit dem systematischen Konstruieren eines technischen Produkts, z.B. Brücke, Auto, CD–Player. Man kann allgemein sagen: Durch Konstruktion eines Programms wird aus der Universalmaschine Rechnersystem eine Spezialmaschine, beispielsweise zur Auswertung von Meßreihen oder zur Temperaturregelung in einem Reaktor.

Wer Maschinen konstruiert, z.B. Roboter oder Lenkwaffen, muß darauf gefaßt sein, daß er für die Auswirkungen des Einsatzes seiner Maschinen mit verantwortlich gemacht wird. Das gilt für Programme um so mehr, als sich solche rechnergestützten Spezialmaschinen ungemein leicht ausweiten und miteinander verknüpfen lassen.

Desweiteren gibt es Programme und Programmsysteme, z.B. zur Büroautomatisierung oder zur Optimierung des öffentlichen Personenverkehrs, die als Spezialmaschinen nur ungenügend zu beschreiben sind. Die Konstruktion solcher Programme hat nicht nur Auswirkungen auf Arbeitsplätze, sondern greift unmittelbar in Arbeitsabläufe und zwischenmenschliche Kontakte ein. Beim Konstruieren solcher Programme treten also auch vielfältige organisatorische, arbeitsrechtliche, soziale oder politische Fragen auf.

2.3 Beispiel einer Problemanalyse

Vor der Behandlung eines konkreten Beispieles soll kurz auf einige generelle Aspekte der Problemanalyse aufmerksam gemacht werden.

Möchte man die Lösung eines Problems mit Hilfe des Computers erleichtern, so muß man sich als erstes fragen, ob das **Problem in geeigneter Weise quantifizierbar** ist. Quantifizierbare Größen zeichnen sich durch die Existenz einer Einheit (Euro, Stück, kg, ...) aus, nur sie lassen sich durch Rechenoperationen weiterverarbeiten. Eng verknüpft mit dem Problem der Quantifizierung ist die Frage nach der **Güte und Auswahl der Eingabedaten**. Nur mit eindeutig definierten Grenzen und Geltungsbereichen sind Ergebnisse aussagekräftig. Ein populäres Beispiel hierfür sind die unter dem Namen „Ökobilanzen“ bekannten Umweltverträglichkeitsprüfungen, bei denen man durch unterschiedliche Grenzziehung zu vollkommen konträren Aussagen kommen kann. Über die Genauigkeit der Eingangswerte sollte man sich ausreichend Klarheit verschaffen, macht doch ein Ergebnis, welches genauer als die Eingabedaten ist, keinen Sinn. Allgemein gilt der wichtige Grundsatz:

Der Output eines Programms ist höchstens so genau wie sein Input.

Ein Programm erzeugt somit keine Information, es transformiert sie nur. Die **Transformationsvorschrift** muß aus sorgfältigen theoretischen Überlegungen hergeleitet werden. Es kann eine Bilanzierungsart oder eine daraus gewonnene Rechenvorschrift sein. Die **Ausgabedaten** schließlich müssen die in der Problemstellung aufgeworfenen Fragestellungen ausreichend beantworten. Hierbei ist es wichtig, die Ausgabedaten in ihrem — durch die Eingabedaten limitierten — Geltungsbereich zu bewerten. Ebenso sind Fehlermeldungen für die beim Rechenprozeß auftretenden Fehlerfälle vorzusehen. Als Beispiel können hier physikalisch sinnlose Zwischenwerte (negative Volumina etc.) genannt werden, welche eventuell im Endergebnis nicht sichtbar auftreten, es aber vollkommen verfälschen. Zusammenfassend sollte man also folgende Punkte beachten:

- Welche Daten gehen in das Programm ein und in welchen Grenzen? Insbesondere: Welche Daten sind nicht als Eingabe akzeptabel?
- Nach welchem Verfahren wird das Ergebnis berechnet — Algorithmus?
- Welches sind die erwarteten Ausgabedaten? Insbesondere: Welche Fehlerfälle können auftreten?

Nun zu unserem konkreten Beispiel:

Viele Studenten und Studentinnen sind ganz oder zum Teil auf finanzielle Unterstützung durch die Eltern angewiesen. Oftmals wird die elterliche Zahlung am Beginn des Studiums festgelegt und orientiert sich entweder direkt an einer für angemessen erachteten Kaufkraft oder indirekt am geltenden Bafögsatz. Sehr problematisch ist nun, daß jährliche Erhöhungen im allgemeinen nicht vorgesehen sind. Bei einer Geldentwertung von 4,5% p.a. erleidet man über einen Zeitraum von zwei Jahren einen Kaufkraftverlust von rund 8,8%! Da es Eltern gibt, die dieser Problematik eher verschlossen gegenüberstehen, soll es uns ein Programm ermöglichen, stichhaltige Argumente zur Lockerung der zögerlichen Zahlungsmoral aufzubieten.

Aufgrund der Zweideutigkeit des Wortes „Ausgabe“, einmal als finanzieller und einmal als informationstechnischer Terminus, werden im folgenden die Ein- und Ausgabedaten des Programms als „Input“ und „Output“ bezeichnet oder besonders gekennzeichnet.

Das Programm soll monatliche Bilanzen mit Ein- und Ausgabeseite erstellen können. Der Input wird dem Kontoauszug entnommen. Die Bilanzen sollen hinsichtlich der Berücksichtigung der Kosten und Einnahmen flexibel sein. Nicht regelmäßige Posten wie z.B. die Urlaubsfahrt oder die weihnachtliche Zuwendung der Oma sollen im Bedarfsfall nicht mitgerechnet werden. Die monatlichen Bilanzen machen eine Aussage über den Kaufkraftverlust, welcher sich in vier Erscheinungen bemerkbar machen kann:

- Steigende Differenz zwischen Einnahme- und Ausgabeseite. Mithin Anhäufung eines Schuldenberges.
- Keine steigende Differenz zwischen Einnahme- und Ausgabeseite. Ausgleich der Differenz durch zunehmendes Zurückgreifen auf Erspartes oder Sonderzuwendungen (Weihnachten etc.).
- Keine steigende Differenz zwischen Einnahme- und Ausgabeseite. Ausgleich der Differenz durch zunehmenden Anteil eigenen Arbeitslohnes. Hier ist die monatliche Arbeitsstundenzahl eine wichtige Kenngröße der zusätzlichen Belastung.
- Keine steigende Differenz zwischen Einnahme- und Ausgabeseite. Ausgleich der Differenz durch Einschränkung des Konsumverhaltens. Hier ist interessant, ob an den großen Posten oder an den alltäglichen Ausgaben gespart wird.

Zur Analyse dieser Erscheinungen werden im Input–Teil des Programms verschiedene Einnahmen- und Ausgabenkategorien geschaffen. Die Kombination einzelner Kategorien in Verbindung mit einer einfachen Rechenvorschrift erzeugt die gewünschten, aussagekräftigen Daten. Näheres dazu im Pflichtenheft.

2.4 Anforderungsdefinition (Pflichtenheft)

Als Ergebnis der Problemanalyse wird eine möglichst vollständige Anforderungsdefinition (engl. *requirement specification*) formuliert. Sie legt den Leistungsumfang des zu erstellenden Programms fest und kann daher als Vertragsgrundlage für ein projektiertes Programm dienen. Andere Bezeichnungen für diese Anforderungsdefinition sind: funktionale Spezifikation, Pflichtenheft.

Für unser Beispiel sieht dies so aus:

Das Programm gliedert sich in einen Eingabeteil (Input) und einen Berechnungs- und Ausgabeteil (Output). Nach einem Programmaufruf hat man in einem ersten Menü die Wahl zwischen diesen beiden Optionen.

Im **Eingabeteil** werden die einfach vom Kontoauszug abgelesenen Einnahmen und Ausgaben mit Betrag und Buchungsdatum in verschiedenen Kategorien eingegeben und in entsprechenden Dateien gespeichert.

- Kategorien für die Einnahmen

1. Regelmäßige elterliche Zuwendung, Bafög, Wohngeld, Sonstiges
2. Gehaltsszahlung aus eigener Arbeit sowie die dazu nötigen Arbeitsstunden
3. Besondere Zuwendungen (Oma, Tante, Weihnachten, Geburtstag), Liquidation von Sparguthaben

- Kategorien für die Ausgaben

1. Festkosten (Miete, Bewag/Gasag/Kohlen, Telefon, Verein, Versicherungen, Kfz)
2. Barabhebung
3. Große Posten (Reise, Stereoanlage, Gold und Diamanten)

Es ist somit ein weiteres Menü für die Auswahl der Optionen „Einnahmen“ und „Ausgaben“ zu erstellen. Die Untergliederung nach den Kategorien wird durch eine entsprechende Maske erleichtert („Maske“ nennt man ein Bild auf dem Bildschirm, in welchem die gewünschten Eingaben in gekennzeichnete Felder eingetragen werden können. Sie ist so etwas wie ein elektronisches Formular).

Im **Berechnungs- und Ausgabeteil** (Output) sollen fünf verschiedene Möglichkeiten zur Auswahl stehen. Von diesen fünf generellen Optionen werden die ersten beiden nochmal in fünf Einzelbilanzen unterteilt. Diese Einzelbilanzen lassen zum einen eine getrennte Betrachtung der verschiedenen Input–Kategorien für Einahmen und Ausgaben zu, zum anderen bieten sie einen Gesamtüberblick und beinhalten in Punkt (b) noch die geforderte Flexibilität bezüglich der großen Einnahme- und Ausgabeposten. Sämtliche Optionen sind durch entsprechende Menüs abrufbar. Der Bilanzzeitraum soll vorher vom Programm abgefragt werden und ist mit Tag, Monat und Jahr einzugeben. Im einzelnen:

1. Monatliche oder jährliche Einnahmenbilanz
   (a) Gesamt, mit besonderen Zuwendungen
   (b) Gesamt, wahlweise mit oder ohne besondere Zuwendungen, wobei die Zuwendungen einzeln hinzugefügt oder weggelassen werden können. Dazu soll eine Liste über die einzelnen Zuwendungen in dem betrachteten Bilanzzeitraum informieren. Durch Eingabe der Listennummer kann die Position ausgewählt werden.
   (c) Nur Einnahmekategorie 1
   (d) Nur Einnahmekategorie 2
   (e) Nur Einnahmekategorie 3
2. Monatliche oder jährliche Ausgabenbilanz
   (a) Gesamt, mit den großen Posten
   (b) Gesamt, ohne die großen Posten, wobei für die Einzelposten das gleiche gelten soll wie für die besonderen Zuwendungen in Punkt 1.
   (c) Nur Ausgabekategorie 1
   (d) Nur Ausgabekategorie 2
   (e) Nur Ausgabekategorie 3
3. Monatliche oder jährliche Differenz aus sämtlichen Einnahmen und Ausgaben, das heißt aus Punkt 1.(a) und Punkt 2.(a). Hieran läßt sich eine steigende, stagnierende oder sinkende Differenz der gesamten Transaktionen feststellen. Zusammen mit den Bilanzen der einzelnen Einnahme- und Ausgabekategorien lassen sich detailliertere Aussagen über die Ursachen der festgestellten Tendenz machen. Weist z.B. die Ausgabenkategorie 3 sinkende Tendenz auf, so wurde an den großen Posten gespart. Weist gleichzeitig die Einnahmekategorie 2 eine Steigerung auf, so wurde mehr Geld durch eigene Arbeit verdient.

4. Monatliche oder jährliche Bilanz aus beliebigen Einzelbilanzen der Punkte 1 und 2. Dabei sollen die Einzelbilanzen der Einnahme- und Ausgabenseite aus einer Liste durch Wahl der Listennummer ausgewählt werden können. Interessiert man sich beispielsweise nur für die Differenz aus den regelmäßigen Einnahmen und den festen Kosten, so muß die Differenz aus den Einzelbilanzen 1.(c) und 2.(c) gebildet werden.

5. Die Differenzen aus 3 oder 4 sollen in Arbeitsstunden umgerechnet werden. Dazu ist ein, über den Bilanzzeitraum gemittelter, Stundenlohn zu berechnen. Ein Anstieg der monatlichen Arbeitsstunden bei stagnierendem Konsum (Ausgabeneinzelbilanzen 2.(d) und 2.(e)) ist ein sicheres Indiz für den schleichenden Kaufkraftverlust. Zusätzlich zum Mitleidseffekt, den diese Erkenntnis bei den Eltern auslöst, ist ein wichtiges Argument, daß die Arbeitsstunden nicht zum Studieren zur Verfügung stehen und somit das Studium unweigerlich verlängert wird ...

2.5 Programmentwurf

Während die Anforderungsdefinition die Leistungen des Programms beschreibt, kommt es nun darauf an, diese Leistungen in Anweisungen für bestimmte Arbeitsschritte umzusetzen. Eine solche Programmierung heißt imperativ, weil sie auf Anweisungen oder Befehle (Imperativ = Befehlsform) ausgerichtet ist. Dies entspricht den im Ingenieurbereich einer großen Gruppe von Programmiersprachen, wie FORTRAN, Pascal, C usw.

Bei der Entwicklung der Anweisungen (Befehle) werden wir nach der Methode der schrittweisen Verfeinerung (top–down Programmierung) vorgehen: Zuerst wird eine grobe, aber einigermaßen genau umrissene Anweisung aufgeschrieben, wie z.B. „Eingabewerte lesen“ oder „Tabelle drucken“. In einem nachfolgenden Arbeitsschritt werden solche groben Anweisungen verfeinert, d.h. in detaillierte Anweisungen umgeschrieben.

Bei der Verfeinerungsarbeit kommen wir typischerweise immer wieder zu folgenden zwei Problemen:

- Wir müssen Entscheidungen fällen, für die unsere Aufgabenstellung nur wenig oder nichts Konkretes für die Umsetzung hergibt.
- Wir erkennen eine mögliche Fehlersituation und müssen uns die Reaktion des Programms überlegen, z.B. was passiert, wenn eine zu druckende Tabelle keinen einzigen Wert enthält.

In beiden Fällen sollen die Ergebnisse unserer Entscheidungen und Überlegungen aufgeschrieben, d.h. dokumentiert, werden. Die Verfeinerung des Programmentwurfs muß vielleicht noch einmal gesteigert werden, bevor eine Ebene erreicht ist, die gleichsam mechanisch in eine Programmiersprache umgesetzt werden kann. Natürlich hängt es auch von der Erfahrung des Programmierenden ab, wie viele Male verfeinert werden muß. Beim Programmentwurf sollte man einige Punkte bedenken, die für den Anweisungsentwurf wichtig werden:

- Zerlegung des Programms in einzelne Module (Unterprogramme) zur besseren Wartung des Programms.
- Effizienz- und Portabilitäts–Gesichtspunkte. Beide widersprechen sich oft, da Programme oft mit maschinenspezifischen und daher nicht portablen Techniken effizient gemacht werden.
- Verwendung von Methodenbanken (Unterprogrammbibliotheken, Algorithmensammlungen).

Je komplexer die entwickelten Programme werden, um so wichtiger wird die Information über Fehlerfälle. Beim Taschenrechner können wir (nicht sehr gut) mit der einzigen Fehlermeldung -E- leben. Dagegen

hat sich lebensgefährlich ausgewirkt, daß der Steuerrechner des Kernkraftwerks *Three Miles Island* auf Fehlbedienung nur mit -?- zu reagieren wußte. Für umfangreiche Programme sind alle möglichen Fehlersituationen, die das Programm erkennt, auch zu testen. Zuvor werden die dokumentierten Überlegungen und Entscheidungen in eine Teststrategie umgesetzt.

Je größer das Programm, um so wichtiger auch das Benutzerhandbuch. Dieses stellt die Leistung des Programms für eine bestimmte Benutzergruppe dar und beschreibt insbesondere die Ein- und Ausgabe.

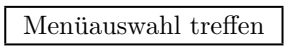
N.B. Auch kurze Programme sollen mit Blick auf Fehlerbehandlung und Benutzerfreundlichkeit entworfen werden.

2.6 Anweisungsentwurf mit Struktogrammen

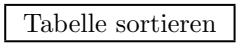
Beim Entwurf von Anweisungen wird mit einem Grobentwurf angefangen, der dann verfeinert wird. Das Ziel ist, Präzision zu gewinnen und die Übersicht nicht zu verlieren. Dazu dienen Entwurfssprachen. Wir verwenden hier Struktogramme als einfache grafische Entwurfssprache.

2.6.1 Grundelemente

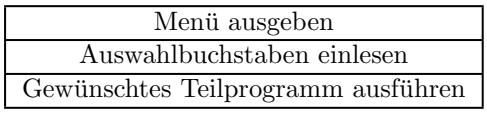
Grundelemente von Struktogrammen sind zunächst rechteckige Kästen; ein solcher Kasten kann eine einfache („grobe“) Anweisung aufnehmen, z.B.



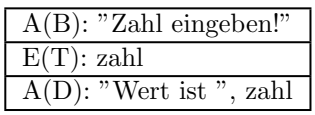
oder



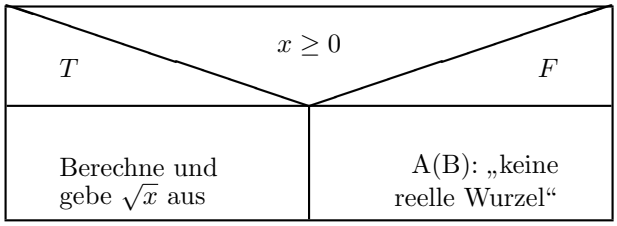
Die rechteckigen Kästchen werden in der Reihenfolge der Abarbeitung so aneinander gefügt, daß wieder ein rechteckiger Kasten entsteht:

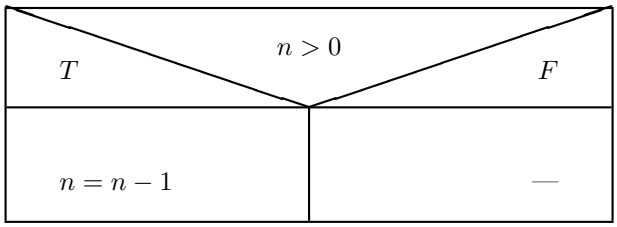


Für Ein- und Ausgaben von Daten verwendet man Abkürzungen, die auch Informationen darüber enthalten, welches Ausagbemedium gewählt werden soll. Die Eingabe wird durch ein großes `E` gekennzeichnet, wobei die Angabe von `E(T)` für die Eingabe von Tastatur steht und `E(D)` für das Einlesen von Daten aus einer Datei. Entsprechend ist `A(B)` die Ausgabe auf dem Bildschirm und `A(D)` die Ausgabe in eine Datei.



Innerhalb eines einfachen Kastens kann eine Strukturierung vorgenommen werden. Außer der vergleichsweise trivialen Zerlegung eines einfachen Kastens in mehrere aufeinanderfolgende Kästen, d.h. Bildung eines Blockes oder einer Sequenz, haben wir als Strukturierungsart Fallunterscheidungen und Schleifen. Allgemein sprechen wir von Ablaufstrukturen des Programms. Dafür hat sich auch die Bezeichnung Kontrollstrukturen eingebürgert (von engl. *control* = steuern; man müßte also eigentlich Steuerstrukturen sagen).





2.6.2 Fallunterscheidung

Zu den Fallunterscheidungen zählt die Verzweigung. Eine Bedingung ist auszuwerten; das Ergebnis ist entweder wahr (true, T) oder falsch (false, F). Dadurch ist bestimmt, welcher von zwei möglichen Zweigen des Programmpfades durchlaufen werden soll. Beispielsweise wird

wenn x größer oder gleich 0 ist bestimme und gebe die Wurzel von x aus

als **Struktogramm** verfeinert zu

und in einer Programmiersprache z.B. zu:

```
IF (x ≥ 0) THEN
   PRINT*, sqrt(x)
ELSE
   PRINT*, 'keine reelle Wurzel'
ENDIF
```

Es kann sein, daß ein Programmpfad leer bleibt, z.B.

verringere positives n um 1

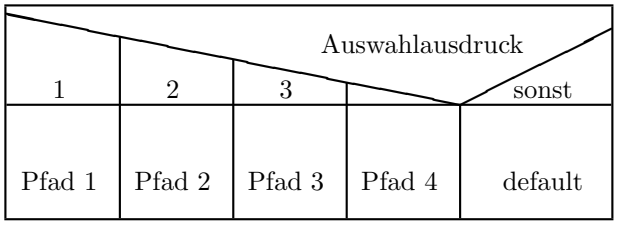
wird als **Struktogramm** zu

und in einer Programmiersprache z.B. zu:

$$\text{IF } (n > 0)\ n = n - 1$$

Man beachte, daß die mathematisch unsinnige Formulierung $n = n - 1$ in einer Programmiersprache eine andere Bedeutung hat: die Anweisung $n = n - 1$ bedeutet, daß in der Variablen n nach Auswertung des Ausdrucks der Wert $n - 1$ abgespeichert wird.

Eine **allgemeine Fallunterscheidung** wertet einen Auswahlausdruck aus, und dieser Wert bestimmt, welcher von mehreren möglichen Zweigen des Programmpfades durchlaufen werden soll.



Dafür gibt es in Programmiersprachen unterschiedliche Umsetzungen, z.B. CASE–Konstruktionen oder geschachtelte Verzweigungen.

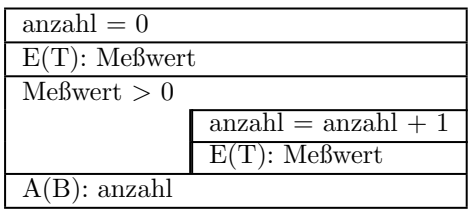
2.6.3 Schleifen

Als Schleifen bezeichnen wir die Wiederholungsanweisungen. Das sind Anweisungen, bei denen durch gewisse Bedingungen gesteuert wird, wie oft eine Anweisung ausgeführt wird. Schleifen gehören zu den wichtigsten Konstrukten der imperativen Programmierung. Sie nutzen die Fähigkeit des Computers, gegebene Anweisungen schnell und zuverlässig millionenfach zu wiederholen. Schleifen bergen aber auch das Risiko, Fehler zu enthalten, deren Ursachen schwierig zu entdecken sind. Wir besprechen zuerst die drei wichtigsten Schleifenarten und danach Vorsichtsmaßregeln beim Programmieren von Schleifen.

Die **Schleife mit Eintrittsprüfung** (kopfgesteuerte oder abweisende Schleife, pre–checked loop) prüft zunächst die Eintrittsbedingung. Das Ergebnis der Prüfung entscheidet, ob die nachfolgenden Anweisungen des Schleifenkörpers (engl. *body*) ausgeführt oder übersprungen werden soll. Nach einem Ausführen des Schleifenkörpers erfolgt ein Rücksprung zur Überprüfung der Eintrittsbedingung, die jetzt als Fortsetzungsbedingung wirkt. Der Körper der Schleife kann aus einer einzigen oder auch aus mehreren Anweisungen bestehen. Die Überprüfung vor jeder Ausführung bedingt, daß der Schleifenkörper möglicherweise übersprungen, also kein einziges Mal ausgeführt wird. Zu diesem Schleifentyp zählen die Zähl– und DO WHILE–Schleifen im Fortran95–Standard (siehe Kap. 4).

Beispiel: Lesen und Zählen von positiven Meßwerten bis zum ersten nicht positiven Wert (Schlußsignal).

Als **Struktogramm** schreibt man solche Schleifen:

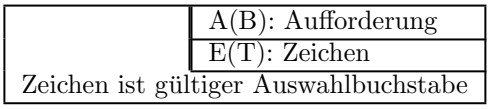


Der linke freibleibende Streifen deutet den Rücksprung zur Überprüfung der Eintrittsbedingung oder das mögliche Überspringen des Schleifenkörpers an.

Bei der **Schleife mit Austrittsprüfung** (endgesteuerte oder nichtabweisende Schleife, post–checked loop) wird zuerst der Schleifenkörper ausgeführt, danach die Austritts- oder Abbruchbedingung überprüft. Das Ergebnis der Prüfung entscheidet, ob die Schleife verlassen wird oder ein Rücksprung zum Beginn des Schleifenkörpers erfolgt. Die Überprüfung am Ende der Schleife bedingt, daß der Schleifenkörper in jedem Fall mindestens einmal ausgeführt wird.

Beispiel: Die Anweisung

läßt sich als **Struktogramm** verfeinern zu



Die **Zählschleife** ist eine besondere Form der Schleife mit Eintrittsprüfung. Die Bedingung ist durch einen Zählbereich gegeben, den eine Lauf- oder Zählvariable systematisch durchlaufen soll. Für jeden gültigen Wert des Zählbereichs soll der Schleifenkörper einmal ausgeführt werden. Zählbereiche sind zunächst einmal Bereiche der ganzen Zahlen, beispielsweise „von 1 bis 3“ oder „von -5 bis -1“, wobei der Zuwachs ohne weitere Angabe +1 ist. Der Zuwachs kann aber beliebige (auch negative) ganzzahlige Werte annehmen, z.B. „von -6 bis +2 mit Zuwachs +2“.

Andere Zählbereiche oder aufzählbare Bereiche hängen von der verwendeten Programmiersprache ab. So ist in Pascal „die Buchstaben von A bis Z“ möglich. In Fortran95 können jedoch nur ganze Zahlen verwendet werden. Der Zuwachs ist stets +1, falls nichts anderes angegeben wird; er kann auch eine reelle Zahl sein: „von 2.47 bis -13.6 mit Zuwachs -4.227“ .

Das Vorzeichen des Zuwachses ist wichtig, da es anzeigt, ob der Schleifenzähler aufwärts (Zuwachs positiv) oder abwärts (Zuwachs negativ) durchlaufen wird. Steht nämlich „von 1 bis n“ da, und n ist null, so brauchen wir die Klarstellung, ob es sich um „1 bis n mit Zuwachs -1“ handelt — dann wird der Schleifenkörper zweimal ausgeführt oder um „1 bis n mit Zuwachs +1“ — dann wird der Schleifenkörper übersprungen.

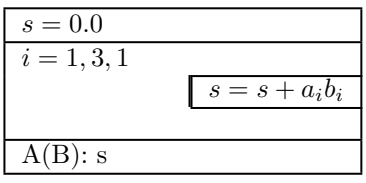
Der Zählbereich wird für die Laufvariable i, wobei „i von n bis m mit Zuwachs k“ gehen soll, als

$$i = n, m, k$$

geschrieben.

Wie bereits erwähnt, sind solche Schleifen in Fortran95 Schleifen mit Eintrittsbedingung, da der Zählindex vor den Durchlaufen der Zählschleife überprüft wird.

Beispiel: Berechnen und Drucken des Skalarproduktes s der beiden Vektoren $a = (a_1, a_2, a_3)$ und $b = (b_1, b_2, b_3)$ nach der Formel $s = \Sigma_i a_i b_i$.



Die **Vorsichtsmaßnahmen** beim Programmieren von Schleifen sollen helfen, die folgenden simplen Fragen sicher zu beantworten:

- Komme ich richtig in die Schleife hinein (Initialisierung)?
- Liefert die Schleife das gewünschte Ergebnis (Korrektheit)?
- Komme ich sicher aus der Schleife heraus (Terminierung)?

Die Wichtigkeit der **Initialisierung** sehen wir an zwei der drei obigen Beispiele: Bevor mit dem Abzählen von Meßwerten begonnen werden kann, muß die Variable `anzahl` auf null gesetzt werden; das Gleiche gilt für die Variable s, bevor die Aufsummierung des Skalarprodukts beginnen kann.

Um die **Korrektheit** einer Schleife zu beweisen, gibt es die Methode der **Programm–Verifikation**. Dabei läßt sich mittels geschickt ausgewählter Zusicherungen (engl. *assertions*) für bestimmte Stellen des Programms zusammen mit den dazwischen liegenden Anweisungen die Korrektheit in der Art eines mathematischen Beweises folgern. Wir können hier auf diese für ein gründliches Informatikstudium unerläßliche Methode nicht eingehen.

In ähnlicher Weise läßt sich auch die Terminierung zeigen. Diese muß eigens bewiesen werden, denn eine Schleife — beispielsweise das Suchen in einer Tabelle — kann das korrekte Ergebnis liefern und dennoch endlos weiterlaufen.

2.6.4 Verfeinerung

Als letztes Struktogrammkonstrukt beschreiben wir die schon vorher erwähnte Verfeinerung. Es kann nützlich sein, durch Einrahmen anzumerken, daß eine bestimmte Anweisung bereits an einer anderen Stelle verfeinert worden ist, z.B.

Der Verfeinerung entspricht in den imperativen Programmiersprachen der Aufruf eines Unterprogramms, auch als Prozedur- oder Subroutine–Aufruf bezeichnet.

Struktogramme erzwingen sauberes, strukturiertes Programmieren, d.h. Programme, die wesentlich leichter les- und wartbar sind. Es gibt allerdings Situationen, in denen von einer klaren Programmstruktur abgewichen werden muß (z.B. Fehlerbehandlung) und die mit Struktogrammen nicht ohne weiteres darstellbar sind. Unabhängig von der verwendeten Entwurfssprache sollte stets größter Wert auf gut strukturierte Programme gelegt werden (siehe auch Abschnitt 2.8).

Die sprachliche Formulierung des Programms geschieht mit einer Programmiersprache, die dem Problem angemessen sein sollte. In Kapitel 4 werden wir bei der Besprechung von Programmiersprachen näher darauf eingehen. Einige Hinweise auf guten Programmierstil findet man schon in Abschnitt 2.8.

2.7 Test, Dokumentation, Wartung

2.7.1 Test

Zum Testen wird das Programm mit den schon während des Programmentwurfes festgelegten Testdaten ausgeführt. Jede Fehlermeldung muß wirklich erzeugt werden. Durch systematisch angelegte Testreihen läßt sich sicherstellen, daß ein Programm für einen Bereich von Eingabewerten korrekt arbeitet. Prinzipiell gilt aber, daß sich durch Testen ein Programm höchstens falsifizieren, d.h. als inkorrekt erweisen läßt. Verifizieren, d.h. als korrekt beweisen, ist nahezu unmöglich, weil das Durchprobieren sämtlicher Kombinationen von Eingabedaten bei ernstzunehmenden Programmen jenseits aller Möglichkeiten liegt.

Die Ergebnisse der einzelnen Tests sind in einem Testbericht einzutragen, der in knapper Form in die Dokumentation eingeht.

2.7.2 Dokumentation

Das Gesamtergebnis der Abwicklung einer Programmieraufgabe ist die Dokumentation. Diese wird mit mindestens zwei verschiedenen Zielen entwickelt. Einerseits dient sie den Anwendern, die das Programm lediglich benutzen wollen. Sie brauchen vor allem das Benutzerhandbuch, das durch Anwendungsbeispiele ergänzt sein sollte. Andererseits dient die Dokumentation bei der Installierung und Pflege oder Weiterentwicklung des Programms den Systementwicklern und Programmierern. Auch dort, wo Systementwickler, Programmierer und Anwender ein und dieselbe Person sind, ist der Wert einer Dokumentation nicht zu unterschätzen. Es ist unglaublich, wieviel Zeit und Mühe es kosten kann, das selbstgeschriebene Programm nach Ablauf eines Jahres zu verstehen.

Für die Zusammenstellung der Dokumentation, die für eine Übungsaufgabe erforderlich ist, hier eine Checkliste:

- **Titelblatt** mit Namen des Programms, Datum, Ort, Namen von Verantwortlichen.
- **Aufgabenstellung** und **funktionale Spezifikation**. Hierbei kann es nützlich sein, Überlegungen und Entscheidungen aus der Problemanalyse mit anzugeben, beispielsweise den Grund, warum ein bestimmtes Berechnungsverfahren nicht verwendet wurde.
- **Struktogramme** und **Vereinbarungen**. Es ist nicht nur der erste Entwurf beizufügen, sondern auch die notwendigen Verfeinerungen.
- **Testvermerk**, bei umfangreichen Programmsystemen die Teststrategie und eine Zusammenfassung des Testberichts. Wichtig ist die Angabe, auf welchem Rechner und unter welchem Betriebssystem das Programm entwickelt und getestet wurde.
- **Programmlisting** und **Ausgabedaten** eines Beispiellaufes.
- Für umfangreiche Programmsysteme ist ein eigenes **Benutzerhandbuch** zu erstellen.

2.7.3 Wartung und Pflege

Die Wartung eines Programmsystems läßt sich mit der Wartung eines technischen Geräts nicht unmittelbar vergleichen. Ein Programm hat keine Teile, die sich abnützen oder verschleißen. Es hat jedoch in der Praxis so gut wie immer Fehler, die erst bei der Benutzung entdeckt werden. Die Behebung solcher Fehler und die Erweiterung des Anwendungsbereichs sind die wichtigsten Wartungsarbeiten. Die Bedeutung einer guten Dokumentation hierfür ist offensichtlich und kann nicht genug betont werden.

Erweiterungen des Anwendungsbereichs ergeben sich einerseits durch Umstellung auf andere Rechner- und Betriebssysteme, andererseits durch Änderungen der Aufgabenstellung. Wenn beispielsweise die gesetzlichen Bestimmungen zur Lohnsteuerberechnung geändert worden sind, muß ein weniger flexibles Programm zur Lohnabrechnung umgeschrieben werden.

2.8 Programmierstil – wie schreibt man ein gutes Programm?

An dieser Stelle bringen wir einige allgemeine Hinweise, die das Schreiben von guten Programmen ermöglichen können. Alle Hinweise sind praxiserprobt und bewährt.

Um ein Programm warten zu können, muß es klar strukturiert und lesbar sein. Die klare Struktur im Großen erreicht man durch **Gliederung** des Programms und **Zerlegung** in kleinere Module (Unterprogramme, Prozeduren). Diese Module lösen genau umrissene, beschränkte Aufgaben und besitzen klar definierte **Schnittstellen**: möglichst nur einen Eingang, einen Ausgang, notwendige Übergabeparameter (Eingabe-, Ausgabeparameter). Für die **Größe** eines Moduls läßt sich kein fester Wert angeben, da sie stark von der zu lösenden Aufgabe beeinflußt wird. Als Faustregel kann ein Umfang bis maximal zwei Seiten Quelltext–Listing (Programmcode) dienen.

Auch die einzelnen Module sollten in sich strukturiert sein. Damit ist einerseits die Struktur im Programmablauf gemeint, andererseits der formale Aufbau des Moduls. Die klare Ablaufstruktur erreicht man durch fleißiges Verwenden von Konstrukten zur **strukturierten Programmierung**: also kein wildes Springen im Programm, möglichst Ablauf von oben nach unten. Der formale Aufbau sollte für alle Module einheitlich sein. Die wichtigen Stichworte hierbei sind **Lesbarkeit** und **Selbst–Dokumentation**. Ein sich selbst dokumentierendes Unterprogramm beginnt mit einem **Programmkopf**, der aus einem **Kommentarteil** und einem **Vereinbarungsteil** besteht, dann folgt der eigentliche Prozedurrumpf. Der **Kommentarteil** umfaßt die Punkte

- **Zweckangabe**: was tut die Prozedur
- **Parameterbeschreibung**: Welche Eingabe-/Ausgabeparameter hat die Prozedur?
  Welchen Typ und welche Struktur haben diese Parameter?
- **Sonstige Hinweise**: z.B. Welche Algorithmen werden verwendet? Welche Grenzen müssen beachtet werden (Genauigkeiten, Dimensionen, ...)?
- **Modifikationsgeschichte**: Wer hat wann was geändert und warum ?

Der **Vereinbarungsteil** beinhaltet Definitionen und Vereinbarungen für die verwendeten Größen sowie Initialisierungen (z.B. Nullsetzen von Variablen).

Der **Prozedurrumpf** beinhaltet das eigentliche Unterprogramm, er besteht aus den Programmanweisungen. Hierfür sollten die folgenden Punkte beachtet werden.

- **Namensgebung**: Namen (von Variablen, Prozeduren, usw.) sollen aussagekräftig sein. Dem Computer ist es egal, welchen Namen man wählt; dem Programmierer, der das Programm warten soll, helfen prägnante Namen ungemein.
- **Optische Trennung**: Teilausdrücke von längeren Anweisungen sollten durch Leerzeichen getrennt werden, z.B. vor und hinter einem Gleichheitszeichen. Blöcke von zusammengehörenden Anweisungen sollten durch Leerzeilen oder leere Kommentarzeilen getrennt werden.
- **Einrücken**: Fortsetzungszeilen sollten eingerückt werden, z.B. um drei Spalten. Einrückungen sind auch angebracht für den Wiederholungsbereich einer Schleife sowie für die zu einem Verzweigungsblock (IF–Block) gehörenden Anweisungen. Einrückungen werden geschachtelt, z.B. wenn Fortsetzungszeilen im Schleifenwiederholungsbereich vorkommen.
- **Kommentare**: Kleinere Anweisungsblöcke, auch einzelne Anweisungen, werden mit beschreibenden Kommentaren versehen, um zu erklären, was sich tut. Die genaueste, am wenigsten teure Dokumentation ist ein Code, der sich selbst dokumentiert!
- **Ein-/Ausgabe–Kommentare**: Eingabeanforderungen des Programmes sollten angeben, welche Größen, gegebenenfalls in welchem Format erwartet werden. Die Programmausgabe soll durch prägnate Kommentare ergänzt werden.

Die bisher gegebenen Hinweise zum Programmierstil sind nahezu unabhängig von der verwendeten Programmiersprache.

2.9 Zusammenfassung

Programmieren ist das Entwerfen einer angemessenen Lösung für ein Problem und deren Umsetzung in Anweisungen an einen Rechner. Wir unterscheiden die Phasen Problemanalyse, Programmentwurf, Programmimplementierung, Test und Abnahme, Integration, Pflege & Wartung.

Das Ergebnis der **Problemanalyse** ist die Definition der Anforderungen an das Programm (Pflichtenheft).

Beim **Programmentwurf** werden die Leistungen des Programms durch schrittweise Verfeinerung von Grobanweisungen in eine Folge von Anweisungen umgesetzt. Außerdem werden die Datentypen und -strukturen vereinbart. Für umfangreiche Programme kommen noch Teststrategie und Benutzerhandbuch hinzu.

Für den Entwurf von Anweisungen verwenden wir die einfache grafische Entwurfssprache der Struktogramme. Die Elemente dieser Sprachen entsprechen den Anweisungen der imperativen Programmierung.

Einfache *Anweisungen* entsprechen rechteckigen Kästen, die eine Beschreibung der jeweiligen Anweisung aufnehmen. Die Aufeinanderfolge solcher Kästen bedeutet die sequentielle Ausführung der enthaltenen Anweisungen als Block.

Bei den *Fallunterscheidungen* unterscheiden wir die Verzweigung und die allgemeine Fallunterscheidung.

Bei den *Schleifen* unterscheiden wir die vor-, die nachprüfende und die Zählschleife. Schleifen sind mächtige Programmierkonstrukte, die besondere Vorsichtsmaßnahmen des Programmierers erforderlich machen, um Initialisierung, Korrektheit und Terminierung zu überprüfen.

Die *Verfeinerung* kann ebenfalls als Strukturierung aufgefaßt werden.

Die **Implementierung** des Programms geschieht mit einer dem Problem angemessenen Programmiersprache. Der Quelltext des Programmes soll lesbar und möglichst *selbstdokumentierend* sein.

Das fertige Programm wird möglichst vollständig **getestet**.

Ein wesentlicher Bestandteil der Programmieraufgabe ist die **Dokumentation** des Programms. Sie ist Basis für die **Wartung** des Programms (Fehlerbeseitigung, Weiterentwicklung).[1]

[1] Teile dieses Kapitels wurden vom Vorlesungsskript von Prof. Biedl (FB13) übernommen und überarbeitet und ergänzt.

2.10 Fragen und Übungen zu Kapitel 2

F 2.1 Programmieren läßt sich in verschiedene Phasen unterteilen. Ordnen Sie die folgenden Begriffe und erklären Sie sie kurz (1 Satz oder Stichworte): Programmentwurf; Test und Abnahme; Programmimplementierung; Wartung; Problemanalyse

F 2.2 Zu welcher Phase gehört die Definition des Pflichtenheftes?

F 2.3 Nennen Sie mindestens 2 gute Gründe, warum es wichtig ist, Programme gut zu dokumentieren.

F 2.4 Wann sollte man Zählschleifen, und wann Schleifen mit Ein- bzw. Austrittsbedingung benutzen?

F 2.5 Schreibe für das folgende Programm ein Struktogramm. Ermittle dann, welchen Output das Programm erzeugen würde.

Programm: Zuerst werden die 3 Variablen i, j, k auf Null gesetzt. Dann wird eine Schleife solange durchlaufen, bis j größer 10 ist. In der Schleife wird zuerst gefragt, ob j größer als k ist. Falls ja, wird k um 2 erhöht. Falls nein, wird k um l erhöht und dann j um den Wert von k erhöht.

Nach der Schleife wird eine Zählschleife mit Zählindex i von 1 bis 10 durchlaufen. In der Zählschleife wird j jeweils um i erhöht.

Zum Schluß sollen die Werte für i, j, k ausgegeben werden.

F 2.6 Welche Probleme sind durch Programme lösbar?

F 2.7 Beschreibe die Methode der schrittweisen Verfeinerung (top down Programmierung).

F 2.8 Die Struktur eines Programms besteht letztlich aus zwei wesentlichen Teilen. Welchen?

2.11 Antworten zu den Fragen und Übungen zu Kapitel 2

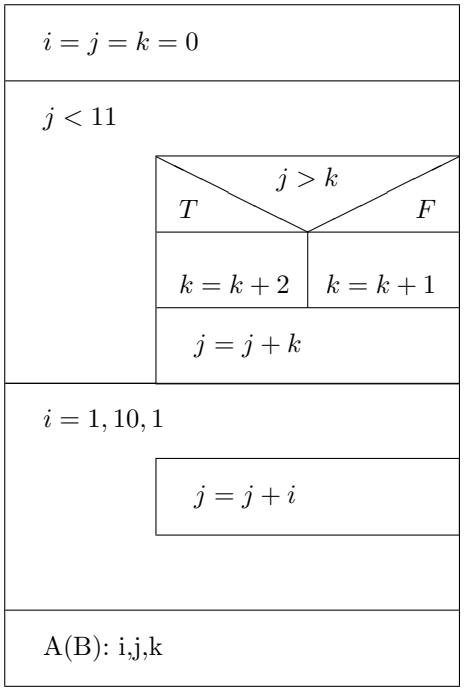
A 2.1 ... siehe Skript ...

A 2.2 Problemanalyse

A 2.3 Wartung
Einfacheres Programmieren
Einfachere Fehlersuche

A 2.4 Zählschleife, wenn Anzahl der Schleifendurchläufe vorher bekannt ist, sonst normale Schleifen.

A 2.5 Output: i=11, j=68, k=6



A 2.6 Jede Aufgabe, deren Vorgaben sich in angemessener Weise in Zahlen darstellen lassen und deren Lösung in endlich vielen Schritten berechnet werden kann.

A 2.7 Zuerst wird der Algorithmus mit wenigen groben Anweisungen beschrieben. Jede dieser groben Anweisungen wird dann durch mehrere detailliertere Anweisungen ersetzt, usw. bis die verfeinerten Anweisungen direkt in einer Programmiersprache formuliert werden können.

A 2.8 Datenstruktur und Befehlsfolge (Algorithmus)

Kapitel 3

Betriebssysteme

Für alle im folgenden aufgeführten Kommandos gilt eine einheitliche Nomenklatur:

- Kommandos gelten i.d.R. für Linux
- optionale Parameter werden in [ ] angegeben

3.1 Aufgabe, Struktur und Dienste eines Betriebssystems

Die Betriebsarten eines Rechners ergeben sich erst aus dem Zusammenwirken der Hardware mit einem Betriebssystem. Ein Betriebssystem ist ein Komplex *problemunabhängiger* Systemprogramme, die den Anwender bei der Nutzung eines Rechners unterstützen, alle auf dem Rechner ablaufenden Prozesse koordinieren und überwachen und dabei für eine möglichst optimale Ausnutzung der Hardware des Systems sorgen.

Dabei ist die Ausführung aller gerätenahen Operationen sowie der Organisationsaufgaben durch das Betriebssystem wichtig. Dem Benutzer wird vom Betriebssystem eine Schnittstelle bereitgestellt, hinter der alle hardware–nahen Prozesse verborgen bleiben.

Umfang und Komplexität der Betriebssysteme haben mit der stürmischen Entwicklung der Informationstechnik und den daran gestellten Anforderungen zugenommen. So wurden Möglichkeiten geschaffen, mehrere Prozesse, gegebenenfalls von mehreren Benutzern (*Multi–User–System*), zeitlich (quasi-)parallel auf einem Rechner auszuführen (*Multi–Tasking–System*), um den Durchsatz und die Ressourcenauslastung zu erhöhen.

Es gibt verschiedene Modi, in denen Abläufe des Betriebssystems ausgeführt werden.

Bei der **Batch–Verarbeitung** (Stapelbetrieb) übergibt der Benutzer die Kontrolle über die Abarbeitung seines Programms dem Betriebssystem. Hierfür muss die zu lösende Aufgabe vollständig gestellt sein. Organisationsprogramme des Betriebssystems steuern die Abarbeitung der in der Eingabe–Warteschlange (Input–Queue) stehenden Programme und die Bereitstellung der Ergebnisse in der Ausgabe–Warteschlange (Output–Queue). CPU–Prozesse (CPU — Central–Processing–Unit) und Input/Output–Operationen (Ein-/Ausgabe) von verschiedenen Prozessen können dabei zeitlich parallel ablaufen. Ein typischer Anwendungsfall ist die rechenintensive Lösung von Differentialgleichungen.

Beim **Dialog–Betrieb** bearbeiten Benutzer von Bildschirm–Terminals aus Programme gleichzeitig am Rechner (Timesharing–Betrieb). Diese Prozesse laufen unter der Kontrolle des Benutzers ab, und die Aufgabenstellung muss nicht vollständig sein. Zusätzliche Verwaltungsfunktionen des Betriebssystems

sorgen für die Aufteilung der Rechnerressourcen bei akzeptablen Reaktionszeiten, z.B. durch zeitzyklische Vergabe an aktive Programme. Ein Beispiel dafür wäre das interaktive Kompilieren von Programmen.

Bei der **Echtzeitverarbeitung** (real–time processing) werden besonders hohe Anforderungen an Verarbeitungs- und Reaktionsgeschwindigkeit gestellt. So muss eine bestimmte (maximale) Reaktions- und Antwortzeit des Betriebssystems auf Anforderungen von außen gewährleistet sein. Echtzeitverarbeitung findet man z.B. bei industriellen Steuerungs- und Regelungsprozessen. Ein bekanntes Anwendungsbeispiel ist die Steuerung des Anti–Blockier–Systems von Autos.

Die Stellung und die Funktion eines Betriebssystems lassen sich mit einem Schalenmodell verdeutlichen, s. Abbildung 3.1.

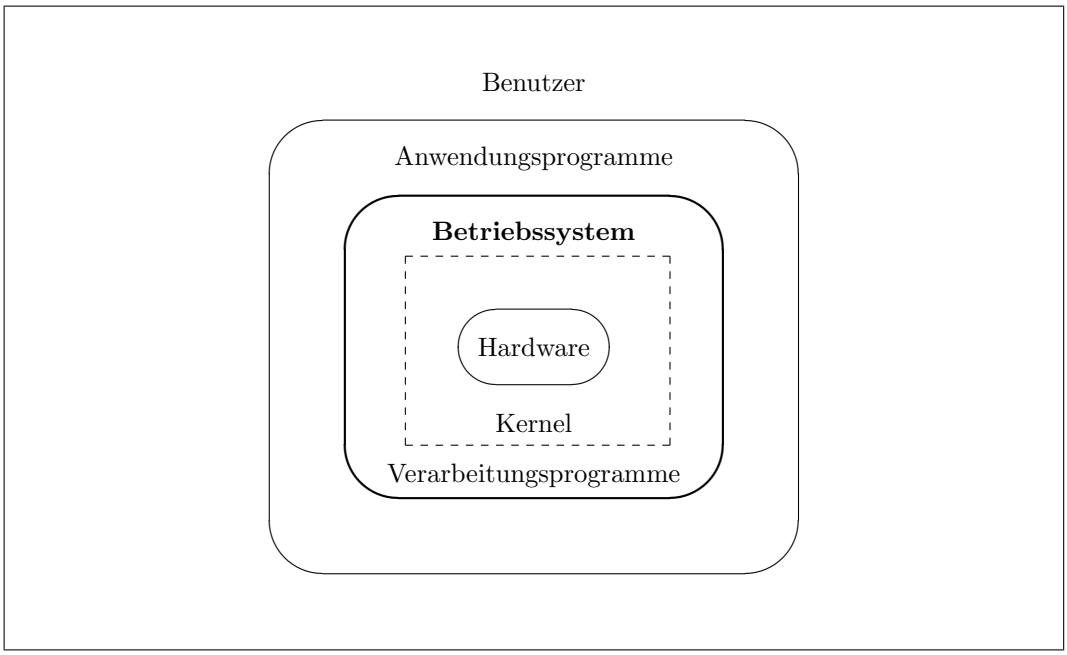


Abbildung 3.1: Schalenmodell eines Betriebssystems

Das Betriebssystem stellt dabei eine Schale dar, die zwischen dem Benutzer und der Hardware des Rechners liegt und die ihrerseits in die beiden Schichten Systemkern (Kernel) und Verarbeitungsprogramme unterteilt ist.

Diese beiden Schichten bauen aus den elementaren Funktionen des Rechners höhere, komplexere Funktionen auf. Diese werden von der obersten Schicht des Betriebssystems dem Benutzer und dem Anwenderprogramm zur Verfügung gestellt.

3.1.1 Der Betriebssystemkern — Kernel

Das Betriebssystem im engeren Sinn, der Betriebssystemkern, umfasst diejenigen problemunabhängigen Komponenten, die gerätenahe Grundfunktionen ausführen:

- Ablaufsteuerung und Verwaltung aller auf dem Rechner laufender Aufträge (engl. *jobs*) und Prozesse für geordnete, konfliktfreie Ausführung (engl. *job control*: Einordnung in Warteschlangen, Vorbereiten zur Ausführung, prioritätsgesteuert starten)

- Ressourcenvergabe an die Prozesse (engl. *task management*): Speicherverwaltung (Vergabe von Speicher an Benutzerprogramme, Zugriffsschutz), Prozessorverwaltung (Vergabe des Prozessors an Benutzerprogramme, zyklisch oder prioritätsgesteuert, mit/ohne Auslagerung aus dem Hauptspeicher (engl. *swapping*))
- Kommunikation mit dem Benutzer und allgemeine Ein-/Ausgabesteuerung: Zugriff auf Dateien (Zugriffsschutz, Paßwortschutz) und Peripheriegeräten (Tastatur, Bildschirm, Maus, Massenspeicher, Drucker, etc.)
- Verwaltung von Dateien (Anlage von Verwaltungsdateien mit Informationen über Zuordnung von Dateinamen und dem Ort ihrer physischen Speicherung)
- Benutzerverwaltung und Abrechnung der Ressourcennutzung über ein Protokoll (engl. *log*) (Gesamtkostenberechnung für Auftragsbearbeitung: engl. *accounting*)

Aufträge an den Rechner werden auf Betriebssystemebene von einem Operateur oder vom Benutzer in der Betriebssystemsprache oder in einer speziellen Auftragssteuerungssprache (engl. *job control language*) formuliert.

Der Auftrag wird von der Auftragssteuerung (engl. *job scheduler*) in einzelne Prozesse (engl. *tasks*) zerlegt, die von den verschiedenen Ressourcen des Systems ausführbar sind, z.B. Eingabe vom Terminal, Verarbeitungsprozess von der Zentraleinheit, Ausgabe auf dem Drucker.

Die Prozesssteuerung startet und überwacht die Ausführung der Prozesse, die aus einer Warteschlange heraus abgearbeitet werden. Für jeden Prozess ist der Bedarf an Betriebsmitteln festgelegt. Auftretende Fehler werden gemeldet, Ursachen diagnostiziert und ein Abbruch oder eine Korrektur eingeleitet. Während eines Prozesses können Unterbrechungen (engl. *interrupts*) durch Fehler, Benutzereingriffe, interne Signale (z.B. Zeitgeber) oder durch Meldungen von anderen Prozessen erfolgen. Für jeden dieser Fälle greift das Betriebssystem mit einer entsprechenden Unterbrechungsbehandlung ein.

3.1.2 Verarbeitungsprogramme

Die zweite (obere) Betriebssystemschicht bilden die Verarbeitungsprogramme. Sie unterstützen die Erstellung, Übersetzung, Erprobung und Abarbeitung von Anwenderprogrammen, die Dateipflege sowie die Systemverwaltung (Administration).

Typische Komponenten dieser Schicht sind:

- Programmierwerkzeuge, wie Editoren zur Erstellung und Veränderung von Anwenderprogrammen,
- Compiler (Übersetzerprogramme) oder Interpreter, die in Assembler (maschinenorientierte Programmiersprache) oder in einer höheren Programmiersprache formulierte Programme in eine Folge von Maschinenbefehlen übertragen,
- Dienstprogramme, die häufig benötigte problemunabhängige Dienste ausführen, wie Arbeiten mit Dateien (Linken, Laden, Bibliotheksverwaltung, Testhilfen (Debugger), Kopieren, etc.),
- Hilfs- und Auskunftsfunktionen

Sowohl der Umfang der Funktionen des Betriebssystemkerns als auch das Spektrum der Verarbeitungsprogramme ist stark abhängig von der Größe und dem Einsatzprofil des Betriebssystems (s. Abschnitt 3.2 — Unix).

Die Bedienung des Rechners erfolgt zunächst tastatur–/kommando–orientiert. Zur effizienteren und bequemeren Benutzung haben sich komfortable fensterorientierte, menügesteuerte, maus–bedienbare Entwicklungsumgebungen (z.B. Visual J++) und Benutzeroberflächen (X11, Windows, GEM) durchgesetzt.

3.1.3 Betriebssysteme für Mikrorechner

In den Anfängen der Mikrocomputertechnik bestand die Systemsoftware meist nur aus Monitor, Assembler und Lader. Der Monitor (von engl. *to monitor*: überwachen) bot vergleichsweise bescheidene Dienste zur Abarbeitung von Programmen sowie einfache Debugger–Funktionen, wie Darstellung von Register- und Speicherinhalten auf dem Bildschirm.

Die Entwicklung der Hardware zu den heute üblichen 32–Bit bzw. 64–Bit Architekturen ermöglichte und erforderte eine höhere Leistungsfähigkeit von Betriebssystemen. Die Mehrzahl der heutigen Betriebssysteme bietet einen weitaus größeren Funktionsumfang an, als oben beschrieben. Dazu gehören:

- Interpretation von Kommandos
- Unterstützung von Ein-/Ausgabeoperationen (I/O)
- Verwaltung von Dateisystemen
- Prozessverwaltung
- Speichermanagement
- Benutzerverwaltung
- Werkzeuge zur Programmentwicklung
- Steuerung (I/O) vielfältiger Peripheriegeräte (CD–ROM–Laufwerk, Modem, Drucker, etc.)
- Integration von Netzwerken

Man kann wohl davon ausgehen, dass die Entwicklung noch nicht am Ende ist und Hard- sowie Software wechselseitig höhere Anforderungen aneinander stellen. Damit wird auch in Zukunft eine ständige Weiterentwicklung stattfinden.

3.2 Betriebssystem UNIX

3.2.1 Geschichte und Philosophie von Unix

Ein gewisses Hintergrundwissen zur Entstehung von Unix ist hilfreich, um die Funktionsweise von Unix zu verstehen und gewisse Umständlichkeiten von Unix zu akzeptieren.

Die Entwicklung von Unix begann 1969 durch Ken Thompson bei den Bell Labs, einer gemeinsamen Tochter der Firmen AT&T und Western Electric.

Zu dieser Zeit waren die meisten Rechner sogenannte Batchrechner mit Lochkarten- bzw. Lochstreifeneingabe. Ein typischer Arbeitsablauf zur Lösung eines Problems war folgender:

- Aufschreiben des Programms (z.B. in FORTRAN) und der Eingabedaten auf Papier
- Eingabe des Programms am Lochkartenstanzer
- Abgabe des Lochkartenstapels beim Operator
- Abholen der Ergebnisse (normalerweise als Zeilendruckerausdruck) nach einigen Stunden
- Sichten der Ergebnisse

In den Bell Labs wurde zu dieser Zeit an dem moderneren Betriebssystem MULTICS (MULTiplexed Information and Computing System) gearbeitet, das noch stark von dieser Denk- und Arbeitsweise geprägt war.

Ziel von Ken Thompsons Entwicklung war es deshalb, ein System zu schaffen, auf welchem mehrere Programmierer im Team und im Dialog mit dem Rechner arbeiten, Programme entwickeln, korrigieren und dokumentieren konnten, ohne von einem Großrechner mit all seinen Restriktionen abhängig zu sein. Das Ziel war sicherlich nicht die Entwicklung eines Betriebssystems für Homecomputer und PC's, deren „Erfindung“ erst 10 Jahre später erfolgte. Als Wortspiel nannte Thompson dieses Betriebssystem Unics (UNiplexed Information and Computing System), aus dem dann Unix wurde.

Dieses erste Unixsystem lief auf einer PDP–7, einem Kleinrechner der Firma Digital Equipment Corporation (DEC). Die erste Version von Unix war dabei in der Assemblersprache der PDP–7 geschrieben. 1971 wurde Unix auf die PDP–11 (DEC) übertragen. Um bei zukünftigen Projekten die Maschinenabhängigkeit durch die maschinennahe Sprache zu umgehen, entwarf Thompson die Programmiersprache B, aus der dann Dennis Ritchie 1972 die Sprache C entwickelte. Unix wurde dann 1973 in C umgeschrieben und auf der PDP–11 implementiert. Von nun an erfolgte die Weiterentwicklung von Unix in der höheren Programmiersprache C. Im Jahre 1975 wurde die Version 6 fertiggestellt. Diese Version fand starke Verbreitung. Der C–Quellcode für Unix wurde von den Bell Labs für die Kopierkosten an Universitäten abgegeben. Dies hatte für die Computerentwicklung (dramatische) Folgen:

- Aufgrund der geringen Kosten erfolgte eine weite Verbreitung
- Wegen der Formulierung in einer höheren Programmiersprache erfolgte eine Portierung auf viele verschiedene Hardwareplattformen
- Wegen der Formulierung in einer höheren Programmiersprache und des modularen Aufbaus konnten andere Programmierer neue Funktionen zu Unix hinzufügen, was zu einer schnellen Weiterentwicklung des Betriebssystems führte
- Allerdings entstanden damit auch verschiedene Dialekte, die sich aus rechtlichen Gründen nicht direkt Unix nennen dürfen (UNICOS, Ultrix, Linux, XENIX, SINIX etc.)

Ausgangspunkt für viele Derivate von Unix war allerdings die Version 7, welche im Jahre 1979 erschien. Für viele ist diese Version „the last true Unix, an improvement over all preceding and following Unices“.

Von dieser Version stammt auch das Berkeley–Unix BSD ab, das neben der AT&T–Linie die wichtigste Entwicklungslinie von Unix darstellt. Glücklicherweise sind Bestrebungen im Gange, die verschiedenen Entwicklungslinien wieder zur Konvergenz zu bringen. Aus dieser kurzen Historie ergeben sich somit die wesentlichen Vorteile von Unix:

- Multi–User, Multi–Tasking–Konzept
- weite Verbreitung des Systems, deshalb kurze Einarbeitungszeit bei Benutzung neuer Rechner
- großer Leistungsumfang, speziell für Programmentwickler
- preisgünstige Portierung des Betriebssystems auf neue Hardware, da Unix (im wesentlichen) in der Hochsprache C geschrieben ist

Nachteile hat Unix aber auch:

- Zum Teil ist Unix sehr kryptisch, da es ursprünglich für Systemprogrammierer gedacht war.
- Aufgrund des großen Funktionsumfangs verbraucht es im Vergleich zu einfachen Betriebssystemen wie MS–DOS relativ viel Ressourcen.

3.2.2 Rechnerzugang

Unix ist ein multi–user, multi–tasking Betriebssystem. Es muss deshalb zwischen verschiedenen Benutzern unterscheiden können. Dazu erhalten die Benutzer einen sogenannten Benutzernamen (engl. *username*). Die Zugangskontrolle zum Computer erfolgt über ein Passwort (Geheimwort, engl. *password*). Jeder Benutzer (engl. *user*) erhält einen eigenen Bereich auf der Festplatte des Computers, das sog. Heimatverzeichnis (engl. *homedirectory*). Benutzername, Passwort und Heimatverzeichnis bilden die Benutzerzulassung (engl. *account*), die beim Systembesitzer beantragt werden muss. Außerdem gehört jeder Benutzer zu (mindestens) einer Gruppe, s. 3.2.3.4. Eingerichtet wird der Account vom Systemverwalter, der selbst einen speziellen Account auf dem Rechner hat. Sein Benutzername ist *root*. Mit dem Passwort von root hat man totale Kontrolle über einen Unixrechner, s. 3.2.3.4 Teilnehmer der Veranstaltung „Einführung in die Informationstechnik für Ingenieure“ erhalten mit ihrer Anmeldung automatisch eine Zulassung auf dem Micropool des Herrmann–Föttinger–Instituts.

Der physikalische Zugang zu einem Unixrechner kann über die Konsole, angeschlossene Terminals oder über ein Netzwerk erfolgen.

Die **Konsole** besteht aus einer Tastatur und einem Bildschirm, die direkt an den Rechner angeschlossen sind. Jeder Unixrechner besitzt eine Konsole. Bei großen Unixrechnern hat normalerweise nur der Systemverwalter Zugang zur Konsole. Auf dem Bildschirm der Konsole erscheinen z.B. die Systemmeldungen beim Einschalten des Computers.

Besonders bei zentralen Unixrechnern älterer Bauart sind oftmals außer der Konsole noch weitere **Terminals** angeschlossen. Die Terminals haben prinzipiell keine eigene Rechenleistung oder Speicherkapazität, sondern dienen lediglich der Ein- und Ausgabe. Diese „dummen“ Terminals werden zunehmend durch „intelligente“ ersetzt. So kann z.B. ein MS–DOS Rechner mit einem sog. Terminalemulationsprogramm als Terminal betrieben werden.

Schließlich kann der Zugang zu einem Unixrechner über ein **Netzwerk** erfolgen. Dieses Netzwerk kann durch Wählleitungen des öffentlichen Telefonnetzes realisiert sein. Wesentlich häufiger ist die Vernetzung über spezielle Gebäudeverkabelungen z.B. über Koaxialkabel. Es gibt verschiedene Arten und Topologien von Netzwerken, siehe dazu auch Kapitel 9.

In unserem Micropool arbeiten wir mit der grafischen Benutzeroberfläche X, s. Abschnitt 3.2.8. Die Beispiele in diesem Skript gelten für einen Benutzer Karl Hansen. Sein Benutzername ist 'hansen'. Das Einloggen (Computerslang für das Anmelden bei einem Rechner) soll an dem Rechner 'svr50' erfolgen.

```
Welcome to SuSE Linux 8.1 (i386) - Kernel 2.4.19-4GB (tty2).

svr50 login: hansen
Password:
Have a lot of fun...
Last login: Mon Jul 14 11:31:58 from svr50.pi.tu-berlin.de.
You have mail.
Mon Jul 14 15:08:50 CEST 2002
svr50:/home/hansen>
```

Die Eingabe des Passwortes erfolgt unsichtbar. Das vom Systemverwalter vergebene Passwort sollte normalerweise vom Benutzer geändert werden. Dazu wird bei uns das Kommando

```
cfdpasswd
```

benutzt.[1]

```
svr50:/home/hansen> cfdpasswd
```

[1] Auf anderen Unix–Systemen werden gegebenenfalls die Kommandos passwd oder yppasswd benutzt.

```
New password:                    <neues Passwort>
Re-enter new password:           <neues Passwort>
Enter bind password:             <altes Passwort>
Result: Success (0)
svr50:/home/hansen>
```

Wiederum erfolgt die Eingabe der Passwörter unsichtbar. Durch unsere zentrale Benutzerverwaltung gilt auf allen Rechnern im Mikropool der gleiche Account.

Beendet wird eine Sitzung mit dem Kommando

```
exit
```

Das gilt nur, wenn man auf der Konsole gearbeitet hat. In unserem Fall wird meistens mit der grafischen Oberfläche X–Window, s. Abschnitt 3.2.8 auf Seite 55, gearbeitet. Hier hängt es vom jeweils verwendeten Windowmanager ab, wie die Sitzung beendet wird.

3.2.3 Das Dateisystem

3.2.3.1 Dateiarten

Eine Datei (engl. *file*) ist aus Sicht des Benutzers im Allgemeinen eine Folge logisch zusammengehöriger Informationen, aus Sicht des Systems hingegen ist sie zunächst nur eine endliche Folge von Bytes. Eine Datei wird über ihren Namen angesprochen, wobei im Unterschied zu MS–DOS zwischen Groß- und Kleinschreibung unterschieden wird. Intern benutzt Unix sog. Inodenummern (inode–numbers). Unix kennt im wesentlichen fünf Dateiarten:

- normale Dateien (plain files)
- Verzeichnisse (directories)
- Links
- Gerätedateien (special files)
- Pipes (pipes)

Normale Dateien: Eine normale Unixdatei entspricht konventionellen, aus anderen Betriebssystemen bekannten, Dateien. In diesem Sinn kann eine gewöhnliche Datei beliebige Daten aufnehmen, beispielsweise Programme und Texte. Unix erwartet keinerlei spezielle Strukturierung einer Datei in Sätze, Blöcke, Sektoren usw. Vom Standpunkt des Anwenders aus gesehen, hat auch in Unix jede Datei eine interne Struktur; so besteht eine Textdatei beispielsweise aus einer Folge von Zeichen mit einer übergeordneten Zeilenstruktur, wobei die Zeilenenden durch besondere Zeilenendezeichen vermerkt sind.

Verzeichnisse: Verzeichnisse sind Dateien, die Verweise auf weitere Dateien — die auch Verzeichnisse sein können — enthalten. Leere Verzeichnisse enthalten zumindest noch die Verweise auf sich selbst '.' und ihr Elternverzeichnis '..'. Dies hängt mit dem hierarchischen Aufbau des Dateisystems zusammen. Die Anzahl der Einträge ist auf die Anzahl der Inode–Numbers bzw. durch die Größe des Datenträgers limitiert. Üblicherweise stehen genügend Inode–Numbers zur Verfügung, so dass es meist eher Probleme mit der Speicherkapazität gibt.

Links: Unix bietet die Möglichkeit, auf eine Datei mit weiteren Namen zuzugreifen, die sogenannten Links. Es gibt zwei Arten von Links:

- **Hard–Links:** Hier wird unter einem anderen Dateinamen dieselbe Adresse für den Dateiinhalt eingetragen. Danach kann nicht mehr zwischen Original und Linkeintrag unterschieden werden. Bei dieser Art des Links müssen sich die Pfadnamen auf demselben physikalischen Gerät befinden (genauer auf demselben Filesystem). Um den Dateiinhalt zu löschen, müssen sowohl der Originaleintrag als auch der Hard–Link gelöscht werden.
- **Soft–Links:** Hier wird unter dem neuen Dateinamen ein Verweis auf die Originaldatei abgespeichert. Hier unterscheidet sich also der Linkeintrag vom Originaleintrag. Dafür kann der Linkeintrag auch auf einem anderen physikalischen Gerät erfolgen. Der Soft–Link kann mit `rm` wieder entfernt werden. Wird dagegen die Originaldatei gelöscht, so zeigt der Linkeintrag ins „Leere“.

Der Link wird mit dem Kommando

`ln` [`-s`]*originaldatei [linkeintrag]*

hergestellt, wobei die Option `-s` einen Soft–Link erzeugt.

Gerätedateien: Gerätedateien sind Dateien, welche für physikalische Geräte stehen. Durch ihre den normalen Dateien gleiche Behandlung ergibt sich für den Benutzer kein Unterschied zwischen der Ein- und Ausgabe auf Dateien oder auf physikalische Geräte, und damit auch weitestgehende Geräteunabhängigkeit.

Pipes: Pipes sind systeminterne Dateien zur Prozesskommunikation, siehe auch Abschnitt 3.2.6.6 auf Seite 49.

3.2.3.2 Dateibaum

Da ein Unixsystem normalerweise mit mehreren tausend Dateien umgeht, würde der Benutzer den Überblick verlieren, wenn es nicht einen Mechanismus geben würde, der es erlaubt, Dateien zu Gruppen zusammenzufassen. Die Unix–Dateienstruktur ist hierarchisch und wird normalerweise als umgekehrter Baum veranschaulicht. Der Ausgangspunkt eines solchen Baumes ist die Wurzel (root), die unter Unix mit '/' bezeichnet wird. Die Wurzel selbst ist ein Verzeichnis, das sich von anderen Verzeichnissen nur insofern unterscheidet, als dass der Eintrag '..' für das Elternverzeichnis auf sich selbst verweist. Abbildung 3.2 zeigt einen Teil eines typischen Unix–Dateisystems.

Während der gesamten Sitzung ist dem Benutzer zu jedem Zeitpunkt genau ein Verzeichnis zugeordnet, auf das sich seine Kommandos beziehen. Dieses Verzeichnis nennt man das **aktuelle Verzeichnis** (engl. *current working directory*). Der Benutzer kann sich im gesamten Dateibaum bewegen, soweit er dafür die entsprechenden Rechte, s. Abschnitt 3.2.3.4 auf Seite 38, besitzt.

Um sich im Dateibaum zu bewegen, benutzt man das Kommando `cd` (**c**hange **d**irectory). Wichtig ist dabei der Unterschied zwischen relativen und absoluten Pfadnamen. Für jedes Verzeichnis und jede Datei gibt es genau einen absoluten Pfadnamen, der immer im Wurzelverzeichnis (root) des Dateibaumes beginnt, also immer mit '/' anfängt. Der absolute Pfadname der Datei `clean` in Abb. 3.2 lautet z.B. `/home/svr50/hansen/clean`. Beim relativen Pfadnamen beginnt man im aktuellen Verzeichnis. Ist man z.B. im Verzeichnis `/usr/local`, so ist der relative Pfadname der Datei `clean` des Benutzers `hansen` `../../home/svr50/hansen/clean`. Das Kommando `cd` kann mit beiden Möglichkeiten benutzt werden:

`cd` [*absoluter Pfad*]

oder

`cd` [*relativerer Pfad*]

Zur Information, wo man sich zur Zeit befindet, benutzt man das Kommando `pwd` (**p**rint **w**orking **d**irectory).

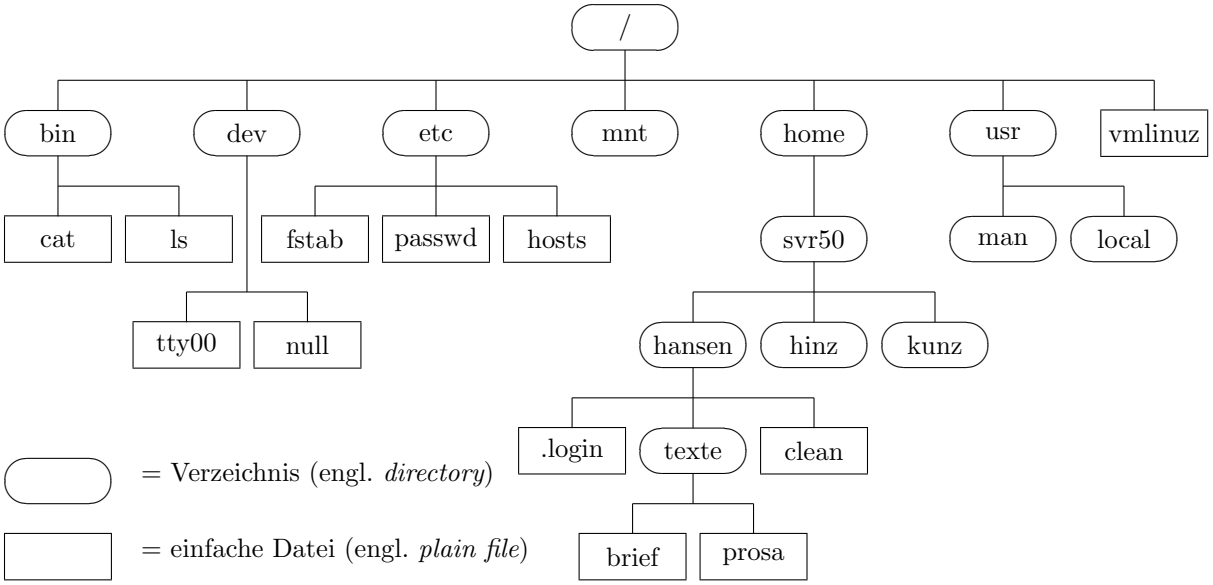


Abbildung 3.2: Beispiel eines Dateibaumes

`pwd`

Oftmals wird das aktuelle Verzeichnis auch an der Eingabeaufforderung (engl. *prompt*) angezeigt. Bei der Eingabe des `cd`–Kommandos ohne Angabe eines Arguments wird in das Heimatverzeichnis (homedirectory) gewechselt. In Abbildung 3.2 ist das Heimatverzeichnis des Benutzers `hansen /home/svr50/hansen`. Die Eingabe von zwei Punkten als Argument des `cd`–Kommandos bewirkt den Wechsel ins Elternverzeichnis des aktuellen Verzeichnisses, also ein Verzeichnis zurück in Richtung Wurzelverzeichnis (/). Beispiel:

```
svr50> pwd
/home/hansen
svr50> cd texte
svr50> pwd
/home/svr50/hansen/texte
svr50> cd ..
svr50> cd ..
svr50> cd ..
svr50> cd ..
svr50> pwd
/
svr50> cd usr/local
svr50> pwd
/usr/local
```

Verzeichnisse werden mit dem Kommando `mkdir` (**m**a**k**e **dir**ectory) erzeugt

`mkdir` *verzeichnis*

und mit `rmdir` (**r**e**m**ove **dir**ectory) entfernt.

`rmdir` *verzeichnis*

Beim Entfernen eines Verzeichnisses muss es leer sein und darf nicht das aktuelle Verzeichnis sein, da das Betriebssystem dann kein current working directory mehr zuordnen kann. Beispiel:

```
svr50> cd
svr50> pwd
/home/svr50/hansen
svr50> mkdir test
svr50> cd test
svr50> pwd
/home/svr50/hansen/test
svr50> cd ..
svr50> rmdir test
svr50>
```

Um sich den Inhalt von Verzeichnissen anzuschauen, gibt es das Kommando

`ls` [`-al`] [*datei ...*]

Dieses Kommando hat sehr viele Optionen, die die Art der Anzeige steuern. Die wichtigsten sind `-l` und `-a`. Die Option `-l` (**l**ong) gibt wichtige Informationen für jede Datei aus. Die Option `-a` (**a**ll) gibt auch die Dateien aus, die mit einem Punkt beginnen. Diese gelten unter Unix als „versteckte“ Dateien, da sie ohne diese Option nicht mit ausgegeben werden. Beispiel:

```
svr50> cd
svr50> pwd
/home/svr50/hansen
svr50> ls
clean  texte
svr50> ls -a
.           .Xauthority  .login      texte
..          .cshrc       .fvwm2rc    clean
svr50> ls -l
total 2
-rw-r--r--  1 hansen edv1         9 Sep 29 15:22 clean
drwxr-xr-x  2 hansen edv1       512 Sep 29 15:26 texte
svr50> ls -la
total 13
drwxr-xr-x  1 hansen edv1       512 Sep 29 15:31 .
drwxr-xr-x  7 root   root       512 Sep 29 15:09 ..
-rw-r--r--  1 hansen edv1      5030 Sep 29 15:09 .Xauthority
-rwxr-xr-x  1 hansen edv1       261 Sep 29 15:09 .cshrc
-rwxr-xr-x  1 hansen edv1       258 Sep 29 15:09 .login
-rwxr--r--  1 hansen edv1        79 Sep 29 15:09 .fvwm2rc
-rw-r--r--  1 hansen edv1         9 Sep 29 15:22 clean
drwxr-xr-x  2 hansen edv1       512 Sep 29 15:26 texte
svr50>
```

Zur Erinnerung: der Eintrag ‘.’ steht für das aktuelle Verzeichnis selber, während ‘..’ für das Elternverzeichnis steht, in diesem Fall also `/home/svr50`.

3.2.3.3 Umgang mit Dateien

Um sich den Inhalt einer Textdatei anzuschauen, gibt es in Unix verschiedene Möglichkeiten. Das gebräuchlichste Kommando ist

`cat` *datei ...*

(con**cat**enate). Bei langen Dateien kann man mit dem Kommando

`more` *datei ...*

oder

`less` *datei ...*

eine seitenweise Ausgabe erreichen. Das Kommando

`head` *datei ...*

zeigt die ersten (10) Zeilen einer Datei und

`tail` *datei ...*

die letzten (10) Zeilen einer Datei an. Außerdem kann man sich Textdateien natürlich mit einem Editor ansehen, siehe Abschnitt 3.2.9 auf Seite 59. Mit dem Kommando

`cp` *alte–datei neue–datei*

(**co**p**y**) erzeugt man eine Kopie einer Datei bzw. kopiert man mehrere Dateien in ein Verzeichnis.

`cp` *alte-datei ... verzeichnis*

Das Kommando `rm` (**rem**ove) entfernt bzw. löscht eine Datei.

`rm` *datei ...*

Achtung: Eine mit **rm** gelöschte Datei ist nicht rekonstruierbar. Es gibt keinen „Papierkorb“, wie z.B. unter Windows.
Mit dem Kommando `mv` (**mo**v**e**) ändert man einen Dateinamen und/oder verschiebt die Datei in ein anderes Verzeichnis.

`mv` *alte–datei neue–datei*

`mv` *alte–datei ... verzeichnis*

```
svr50> cd
svr50> cd texte
svr50> cat prosa
Hoffentlich kommt bald ein Birkenwald,
denn meine Pillen wirken bald.
svr50> cp prosa dirty_prosa
svr50> mv brief brief_an_bollmann
svr50> ls
brief_an_bollmann  dirty_prosa        prosa
svr50> rm dirty_prosa
svr50> ls
brief_an_bollmann  prosa
svr50>
```

Ein weiteres nützliches Kommando ist `file`. Damit erhält man Informationen über den Typ einer (oder mehreren) Datei(en).

`file` *datei ...*

Das ist z.B. nützlich, wenn man drucken, s. 3.2.6.7 auf Seite 49, will und dabei Fehler auftreten.

In unserem Rechnernetz hat jeder Benutzer ein Plattenplatzlimit, d.h. es darf nur eine begrenzte Menge an Daten gespeichert werden. Es ist also notwendig zu wissen, wieviel Plattenplatz belegt ist. Das Kommando

`du` [*verzeichnis*]

ermittelt den genutzten Platz eines Verzeichnisses und seiner Unterverzeichnisse. Angewendet auf das home–directory liefert `du` den aktuell belegten Plattenplatz des Benutzers.

3.2.3.4 Eigentümer, Gruppen und andere Nutzer

Natürlich darf nicht jeder Benutzer eines Unixsystems alle Dateien manipulieren. Die entsprechenden Rechte werden über Attribute der Dateien verwaltet. Wir betrachten dazu die Ausgabe des `ls`–Kommandos mit den Optionen `-lg`. Die Option `-g` bewirkt gewöhnlich die Ausgabe der zugehörigen Gruppe. In unserem Linux ist diese Option schon implizit in der Option `-l` enthalten.

```
svr50> cd
svr50> ls -l
total 2
-rwxr-x---  1 hansen edv1          9 Sep 29 17:20 clean
drwxr-xr-x  2 hansen edv1        512 Oct 29 18:39 texte
svr50>
```

Die Ausgabe des Kommandos `ls -lg` bedeutet im einzelnen folgendes:

- In der ersten Zeile wird nach dem Wort `total` angegeben, wieviele Blöcke die Einträge in diesem Verzeichnis belegen. Ein Block belegt normalerweise 512 Byte. Darauf folgt eine Liste aller Dateien, wobei pro Datei eine Zeile an Informationen ausgegeben wird.
- In der ersten Spalte stehen das Dateiartbit und die 9 Zugriffsrechtebits (protection bits), wobei jedes Bit durch ein druckbares Zeichen dargestellt wird. Das Dateiartbit kann folgende Einträge haben:
  - `d` Datei ist ein Verzeichnis (directory)
  - `l` Datei ist ein symbolischer Linkeintrag (Soft–Link) auf eine andere Datei
  - `-` normale Datei
  - `b` Blockorientierte Gerätedatei
  - `c` Characterorientierte Gerätedatei
  - `p` FIFO–Puffer (named pipe special file)

  Die Zugriffsrechte werden gleich erläutert.
- Die Zahl in der zweiten Spalte gibt die Anzahl der Links auf diese Datei an.
- In der dritten und vierten Spalte stehen der Name des Besitzers und der zugehörigen Gruppe.
- In der fünften Spalte wird die Dateilänge in Bytes angegeben.
- In den nächsten Spalten wird Datum und Uhrzeit der letzten Änderung angegeben. Bei sehr alten Dateien wird statt der Uhrzeit das Jahr angegeben.
- In der letzten Spalte steht der Name der Datei.

Für die Zugriffsrechte muss man die folgenden drei Benutzerklassen unterscheiden:

- u (**u**ser) — der Besitzer einer Datei.
- g (**g**roup) — die Gruppe, der die Datei zugeordnet ist angehört.
- o (**o**ther) — alle anderen Benutzer.

```
-rwxr-xr--   1 hansen edv1    75858 Jun 18 15:13 linux.tex
-rw-rw-r--   1 hansen edv1    64575 Jun 17 13:32 aufgabe.txt
lrwxrwxrwx   1 hansen edv1     1024 May 12 11:38 edv -> /home/stokes/EDV1/
drwxr-x--x   1 hansen edv1    64575 Jun 17 13:32 public
       |||other
    |||group
 |||user
|Dateiart
```

Für jede dieser drei Benutzerklassen werden drei verschiedene Zugriffsrechte gesetzt:

- r (**r**ead) Lesen
- w (**w**rite) Schreiben
- x (e**x**ecute) Ausführen

Insgesamt können also 9 Zugriffsrechte pro Datei gesetzt werden. Diese Zugriffsrechte gibt es für alle Dateien, also auch für Gerätedateien und Verzeichnisse. Mit der Vergabe von Rechten sollte aus Gründen des Datenschutzes und der Datensicherheit sehr vorsichtig umgegangen werden. An der Ausgabe des `ls -l` Kommandos im obigen Beispiel kann man die Zugriffsrechte ablesen. Die ersten drei Bits zeigen die Rechte des Eigentümers, die mittleren die Rechte der Gruppenmitglieder und die letzten drei Bits die Rechte aller anderen Benutzer an. Die Datei `linux.tex` besitzt z.B. als protection mode `rwxr-xr--`. Dies bedeutet, dass der Eigentümer (also hansen) diese Datei lesen, auf ihr schreiben und sie ausführen darf. Alle Gruppenmitglieder der Benutzergruppe edv1 dürfen diese Datei nur lesen oder ausführen. Andere Benutzer dürfen auf diese Datei nur lesend zugreifen.

Die Zugriffsrechte auf Verzeichnisse stellen eine Spezialität dar. Die protection bits sind gleich denen von normalen Dateien, jedoch besitzen die Zugriffsrechte eine etwas andere Bedeutung:

- Um ein Verzeichnis ansehen oder durchsuchen zu können, benötigt man nicht nur das Leserecht für dieses Verzeichnis, sondern auch das Ausführungsrecht. Es müssen also das r– und das x–Bit gesetzt sein.
- Ist das w–Bit eines Verzeichnisses für eine Benutzerklasse gesetzt, so kann jedes Mitglied dieser Klasse eigene Dateien in dieses Verzeichnis eintragen. Eigentümer dieser neuen Dateien ist der Benutzer, der die Datei anlegt. Es ist sehr gefährlich, diese Schreibberechtigung zu vergeben, da jeder, der das Schreibrecht für ein Verzeichnis hat, auch Dateien aus dem Verzeichnis löschen kann, und zwar unabhängig davon, ob er irgendwelche Zugriffsrechte auf diese Datei besitzt oder nicht.
- Wenn jemand Unterverzeichnisse für andere Benutzer zugänglich machen will, muss in allen Verzeichnissen, die in der Hierarchie darüber liegen, mindestens das entsprechende x–Bit gesetzt sein. Beispiel:

  ```
  svr50>pwd
  /home/svr50/hansen/www/
  svr50>ls
  index.html
  ```

Die Datei `index.html` soll für alle (other) lesbar sein. Dann muss in den Verzeichnissen `usr`, `home5`, `hansen` und `www` das x–Bit für „other“ gesetzt sein, und für die Datei `index.html` selbst muss das r–Bit gesetzt sein. Anschaulich gesprochen müssen die Nutzer die übergeordneten Verzeichnisse durchlaufen können und dann das Recht haben, die Datei zu lesen.

Mit dem Kommando `chmod` (**ch**ange **mod**e) kann der Eigentümer einer Datei die Zugriffsrechte dieser Datei ändern. Für dieses Kommando gibt es zwei syntaktisch verschiedene Formen, die aus verschiedenen Unix–„Dialekten“ herrühren:

`chmod` *permissionlist datei ...*

oder

`chmod` *ooo datei ...*

Im ersten Fall gibt der Benutzer eine *permissionlist* und den Namen der Datei an, deren protection mode geändert werden soll, und zwar durch gezieltes Hinzufügen oder Wegnehmen von Zugriffsrechten einzelner Benutzerklassen.

Die *permissionlist* hat dabei folgende Form: *benutzerklasse* ± *rechte*. Als Benutzerklasse ist außer den drei bekannten (u, g und o) noch eine Sammelklasse a (**a**ll) möglich. Beispiel:

```
svr50> ls -l
total 2
-rwxr-xr--  1 hansen edv1         9 Sep 29 15:22 clean
drwxr-xr-x  2 hansen edv1       512 Sep 29 15:38 texte
svr50> chmod o-r clean
svr50> chmod go-rx texte
svr50> ls -l
total 2
-rwxr-x---  1 hansen edv1         9 Sep 29 15:22 clean
drwx------  2 hansen edv1       512 Sep 29 15:38 texte
svr50>
```

In der zweiten Form des Kommandos wird der protection mode als ganzer neu gesetzt. Dabei steht *ooo* für eine dreistellige Oktalzahl, die als Binärzahl, s. Kapitel 5, gelesen genau dort die Einsen hat, wo das entsprechende Zugriffsrecht gesetzt sein soll. Beispiel:

```
svr50> chmod 754 clean
svr50> ls -l
total 2
-rwxr-xr--  1 hansen edv1         9 Sep 29 15:22 clean
drwx------  2 hansen edv1       512 Sep 29 15:38 texte
svr50>
```

Die Oktalzahl 754 lautet in Binärdarstellung `111101100`, was genau den Zugriffsrechten `rwxr-xr--` der Datei clean entspricht.

Bei jeder Erstellung einer Datei werden automatisch vom System voreingestellte protection bits gesetzt. Diese Voreinstellung kann und sollte vom Benutzer bei Sitzungsbeginn auf seine eigenen Bedürfnisse hin mit dem Kommando `umask` (**u**ser **mask**) geändert werden.

`umask` [*ooo*]

ooo ist hierbei wieder eine Oktalzahl, die allerdings etwas anders wirkt als beim Kommando `chmod`. Die Bits, die in der Maske bestimmt sind, werden beim anlegen der Datei gerade *nicht* gesetzt. Man bestimmt also, welche Rechte nicht gewährt werden sollen. Beispiel:

```
svr50> umask 177
svr50> cp clean testdatei
svr50> ls -l
total 3
-rwxr-xr--  1 hansen edv1         9 Sep 29 15:22 clean
-rw-------  1 hansen edv1         9 Sep 29 15:54 testdatei
drwx------  2 hansen edv1       512 Sep 29 15:38 texte
svr50> umask
177
svr50>
```

Jede von diesem Zeitpunkt an erzeugte Datei wird read– und write–Berechtigung für den Eigentümer besitzen, andere Benutzer haben überhaupt keine Zugriffsrechte. Das `umask` Kommando ohne Argument liefert die momentan gültige Einstellung der Maske.

Es sei hier nocheinmal auf den Systemadministrator 'root' verwiesen. Für ihn gelten die Einschränkungen bezüglich der Zugriffsrechte auf Dateien nicht. Er kann sich im gesamten Verzeichnisbaum bewegen, Dateien lesen, verschieben oder auch löschen. Mit dem Passwort für root hat man also umfassende Kontrolle über das System, was zur Aufrechterhaltung eines ungestörten Betriebs des Systems sicher notwendig ist, aber auch eine Missbrauchsgefahr darstellt.

3.2.4 Die Prozessverwaltung

Ein wesentliches Leistungsmerkmal von Unix ist die Fähigkeit zum multi–tasking. Das bedeutet, dass der Rechner quasi–zeitgleich mehrere Programme bzw. Kommandos (tasks) ausführen kann. Bis jetzt wurde in diesem Skript immer ein Kommando gegeben und die Shell, s. Abschnitt 3.2.6 auf Seite 45, antwortete erst nach Beendigung des Kommandos mit einem Prompt. Alle Kommandos wurden also nacheinander abgearbeitet. Wie startet man aber mehrere Programme, die gleichzeitig laufen sollen? Es gibt dazu zwei wichtige Möglichkeiten:

- **Mehrere Terminals:** Bei Benutzung der X–Window Oberfläche, s. Abschnitt 3.2.8 auf Seite 55, kann man mehrere Fenster (Windows) aufmachen. Die Tastatur ist dann exklusiv dem gerade aktiven Fenster zugeordnet. Man hat dann also die Möglichkeit, in jedem Fenster Prozesse zu starten, die dann parallel ablaufen.
- **Hintergrundprozesse:** Durch Anhängen des Zeichens & an ein Kommando wird dieses im Hintergrund abgearbeitet.

  kommando `&`

  Die Shell meldet sich in diesem Fall sofort wieder mit dem Prompt. Der Benutzer erhält auf dem Bildschirm eine Nachricht, wenn der Hintergrundprozess beendet wurde.

In Abbildung 3.3 ist der zeitliche Ablauf von verschiedenen Prozessen dargestellt. Ausgangspunkt ist der Prozess `csh`. Dies ist die C–Shell, also das Programm, das unsere Eingaben annimmt und entsprechende Betriebssystemaufrufe veranlasst. Näheres siehe Abschnitt 3.2.6. In der Abbildung wird im Hintergrund mit `f95` der Fortrancompiler aufgerufen. Da der Compiler im Hintergrund läuft, können sowohl `csh` als auch `f95` aktiv sein, was durch die durchgezogenen Linien dargestellt wird. Dagegen wird das Kommando `ps` im Vordergrund gestartet, so dass der Prozess `csh` bis zur Beendigung von `ps` warten muss, was durch die gestrichelte Linie dargestellt ist. Es soll jetzt erklärt werden, wie man beliebige Prozesse vom Vordergrund in den Hintergrund und zurück schickt. Betrachten wir zunächst die Möglichkeiten, einen Prozess anzuhalten. Mit

`<ctrl+d>`

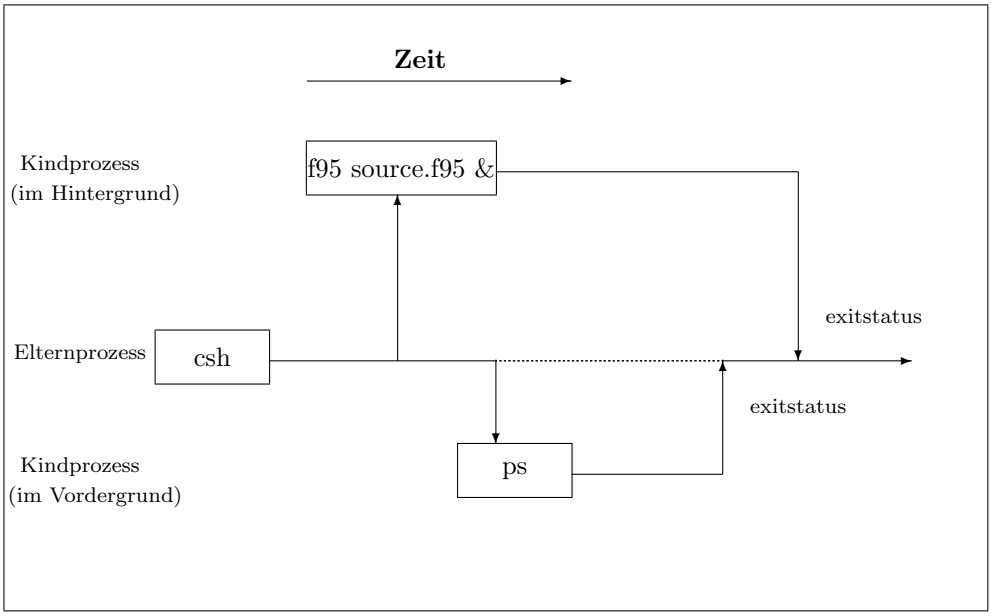


Abbildung 3.3: Zeitlicher Verlauf von Prozessen

wird an einen auf Input wartenden Job ein Dateiendezeichen gesandt. Mit

`<ctrl+c>`

wird an einen Job ein Interrupt–Signal gesendet. Manche Kommandos werden daraufhin (vorzeitig) beendet, andere ignorieren dies. Mit

`<ctrl+z>`

wird ein Kommando nur angehalten, dies aber sicher. Die C–Shell selber verwaltet gestoppte Prozesse und Hintergrundprozesse unter der Vergabe von Jobnummern. Diese kann man mit dem Kommando

`jobs`

auflisten. Mit dem Kommando

`fg [%`*jobnummer*`]`

wird der Job in den Vordergrund geholt und mit

`bg [%`*jobnummer*`]`

in den Hintergrund geschickt. Beispiel:

```
svr50> clean
^Z
Suspended
svr50> jobs
[1]   + Suspended              clean
svr50> bg
[1]    clean &
svr50> jobs
[1]    Running              clean
svr50>
[1]    Done                 clean
```

```
svr50>
```

Das `ps`–Kommando erlaubt die Ausgabe von Systeminformationen über alle laufenden Prozesse.

`ps [-auxl]`

Ohne Optionen gibt `ps` nur die wichtigsten Informationen über die Prozesse aus, die vom Benutzer an diesem Terminal gestartet wurden. In der ersten Spalte wird dabei die eindeutige Nummer jedes Prozesses angezeigt (process identification number — PID). Diese Nummer kann benutzt werden, um mit dem Kommando `kill` Prozesse zu stoppen, egal ob sie im Vorder- oder Hintergrund laufen.

`kill [-9]` *prozeßnummer*

Dabei können natürlich nur eigene Prozesse gestoppt werden und nicht die anderer Benutzer. Die Option `u` des Kommandos `ps` gibt auch die User–ID des Nutzers aus, der den jeweiligen Prozess gestartet hat.

```
svr50> cd
svr50> clean &
[1] 6031
svr50> ps
  PID TT STAT  TIME COMMAND
 5825 p0 S     0:02 -csh (csh)
 6031 p0 S     0:00 /bin/sh clean
 6032 p0 S     0:00 sleep 99
 6033 p0 R     0:01 ps
svr50> kill -9 6031

[1]    Killed               clean
svr50> ps
  PID TT STAT  TIME COMMAND
 5825 p0 S     0:02 -csh (csh)
 6037 p0 R     0:01 ps
svr50>
```

Das Kommando

`top`

zeigt an, welche Prozesse zum jeweiligen Zeitpunkt laufen und wieviel CPU–Zeit sie benötigen. Sollte es bei der Benutzung eines Rechners vorkommen, dass er nicht mit der normalen Geschwindigkeit arbeitet, lohnt der Aufruf des Befehls `top`. Die Ausgabe sieht dann ungefähr so aus:

```
  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
26896 hansen    12   0  5596 5596  3348 R       0 98.3  5.7   0:02 netscape
```

Die Spalte `%CPU` gibt an, dass der `netscape` momentan 98.3% der Rechenzeit beansprucht. Wenn weitere Prozesse auf der CPU rechnen wollen, muss der Scheduler eine Zuteilung vornehmen. Wenn sehr viele Prozesse, viele Ressourcen beanspruchen, können sich die Ausführungszeiten durch mehrfaches Ein- und Auslagern von Prozessen stark erhöhen, was dann zu schlechteren Antwortzeiten des Systems führt. Man sollte dann schauen, ob nicht noch unnötige Prozesse laufen. Das Beispiel des Browsers „Netscape“ ist nicht zufällig gewählt. Es kommt öfter vor, dass dieser Prozess nicht richtig beendet wird. Er kann dann mit dem o.g. `kill` und der entsprechenden PID „abgeschossen“ werden, was, wie schon gesagt, natürlich nur auf einen eigenen Prozess zutrifft.

3.2.5 Das Speichermanagement

Jedes Programm (jeder Prozess), das in einem Rechnersystem läuft, benötigt Speicher, um seine Daten abzulegen. Da in einem Unix–System mehrere Prozesse „gleichzeitg“ laufen können, muss gewährleistet sein, dass es nicht zu gegenseitigen Beeinflussungen kommt. Es gibt in einem Rechnersystem verschiedene Speicherformen, die meist eine hierarchische Ordnung besitzen, s. Abbildung 3.4. Diese gliedert sich nach den Zugriffszeiten. Zusammengefasst ergibt sich daraus ein virtueller Adressraum, der dem Betriebssystem zur Verfügung steht.

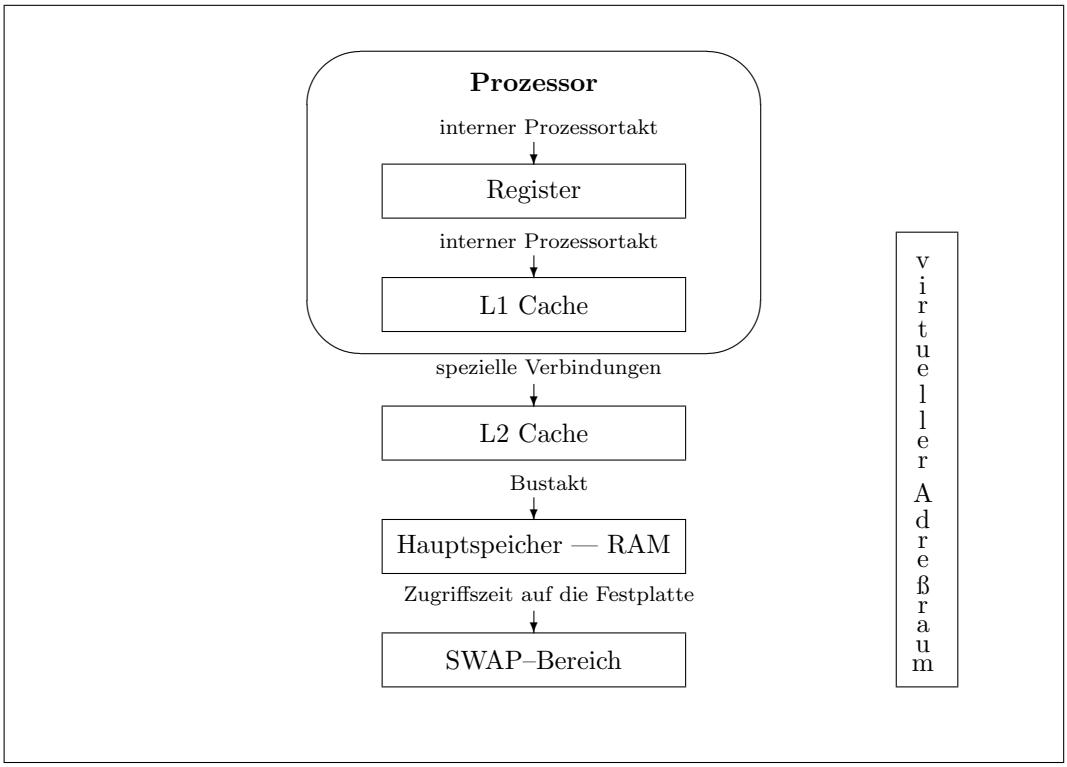


Abbildung 3.4: mögliche Speicherhierarchie eines PCs

Da Speicherbausteine, die eine sehr geringe Zugriffszeit ermöglichen, verhältnismäßig teuer sind, ist auch die Größe hierarchisch. So sind Cache–Bausteine und ihre Ansteuerung wesentlich teurer als RAM (engl. **R***andom* **A***ccess* **M***emory*). Da ein Prozess nicht weiß, an welchen Stellen im Speicher seine Daten zur Laufzeit abgelegt werden, arbeitet er mit virtuellen Adressen, die jeweils bei 0 beginnen. Das Betriebssystem muss mit seiner Speicherverwaltung nun dafür sorgen, dass eine Abbildung der virtuellen Adressen auf die eigentlichen physikalischen Adressen stattfindet. Außerdem muss jedem Prozess ein eigener physikalischer Speicherabschnitt zugeordnet werden, damit es nicht zu Störungen der Prozesse untereinander kommt. Speicherbereiche müssen auch vor unberechtigtem Zugriff geschützt werden. Die Speicherverwaltung wird hardwaremäßig durch die **M**emory **M**anagement **U**nit — MMU unterstützt.

Nun ist es natürlich wünschenswert, dass laufende Programme immer in Speicherbereichen mit geringen Zugriffszeiten liegen. Da aber jede Hierarchiestufe nur begrenzte Kapazitäten hat, muss es Methoden der Zuteilung geben. Wir wollen hier nicht näher darauf eingehen, sondern nur LRU — **L**ast **R**ecently **U**sed als eine Strategie erwähnen. Es wird durch sinnvolles Ein- und Auslagern von Daten versucht, den Prozessor in möglichst hoher Geschwindigkeit mit den Daten zu versorgen. Zu einer signifikanten Verschlechterung der Systemleistung kommt es, wenn mehrere Prozesse mit jeweils hohem Speicherbedarf rechnen wollen und der Hauptspeicher nicht ausreicht. Dann werden Daten auf die Festplatte ausgelagert und müssen von dort regelmäßig neu in den Hauptspeicher geladen werden — swapping.

3.2.6 Die Benutzerschnittstelle — die Shell

Der Benutzer kommuniziert in der Regel nicht direkt mit dem Betriebssystem Unix, da dessen Schnittstellen nach außen zu kompliziert und benutzerunfreundlich sind. Über diesen Schnittstellen liegt eine Schicht, welche die Benutzeranweisungen interpretiert und in Betriebssystemaufrufe umsetzt. Außerdem überprüft diese Schicht die Rückmeldungen des Systems und gibt im Fehlerfall entsprechende Fehlermeldungen aus. Diese Schicht heißt unter Unix Shell, weil sie sich wie eine Schale um den Betriebssystemkern legt. Abbildung 3.5 veranschaulicht das grafisch.

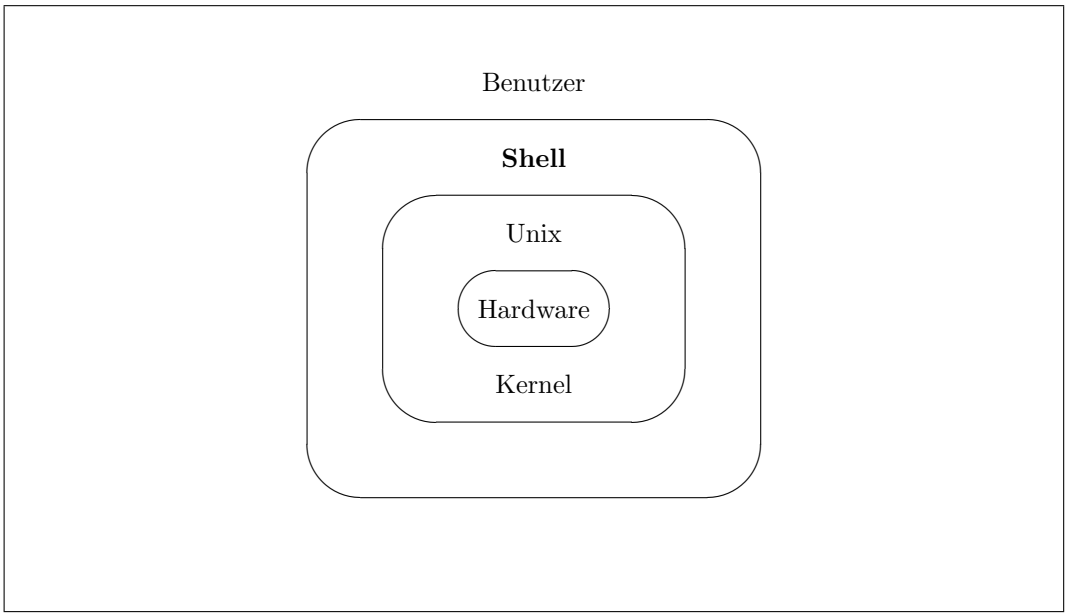


Abbildung 3.5: Die verschiedenen Schichten zwischen Benutzer und Hardware

Die Shell ist kein Bestandteil des Betriebssystemkerns, sondern den Benutzerprogrammen gleichgestellt und leicht austauschbar. Die Shell ist kommandoorientiert und auf die Bedürfnisse eines Programmentwicklers ausgerichtet, d.h. die Kommandosprache ist sehr knapp gehalten und fragt beim Löschen etc. möglicherweise nicht nach. Da die Shell als Programm einfach austauschbar ist, gibt es eine Vielzahl von Varianten:

- `sh`: Die Bourne-Shell ist die Standardshell des Unixsystems
- `csh`: Die C–Shell ist komfortabler als die Bourne–Shell Sie enthält Elemente analog der Syntax der Sprache C
- `ksh`: Die Korn–Shell
- `bash`: Die Born–Again–Shell

In diesem Kurs werden wir die C–Shell benutzen, da sie komfortabler als die Bourne–Shell ist. Es soll deshalb auch nur diese Shell erklärt werden. Neben der einfachen Kommandoausführung stellt die Shell eine ganze Reihe weiterer Konzepte zur Verfügung. Hierzu gehören Umlenkungsmechanismen (3.2.6.4) für Ein- und Ausgabe, Dateinamenexpansion (3.2.6.3) bei Verwendung von Metazeichen sowie programmiersprachenähnliche Kontrollstrukturen.

Die Kommandosyntax ist immer gleich:

kommando–name [*optionen*] [*parameter*]

Dabei ist *kommando–name* der Bezeichner eines Shell–internen Kommandos oder der Dateiname eines Programms. Art und Anzahl der Optionen und Parameter sind programmabhängig. Optionen werden normalerweise mit einem Minuszeichen eingeleitet. Beispiel:

```
svr50> ls -l clean
-rwxr-xr--  1 hansen          9 Sep 29 17:20 clean
svr50>
```

Dabei ist `-l` die Option und `clean` der Parameter. Mehrere Kommandos werden durch Semikolon getrennt

kommando1;*kommando2*;*kommando3*

3.2.6.1 Komfortable Kommandoeingabe

Die C–Shell stellt umfangreiche Möglichkeiten zur komfortablen Kommandoeingabe zur Verfügung. Über den sogenannten History–Mechanismus merkt sich die Shell die letzten Kommandos. Auf den meisten Systemen ist es möglich, diese Kommandos zu editieren, wobei die Syntax der vi–Kommandos, s. Abschnitt 3.2.9 auf Seite 59, benutzt wird. Die Erzeugung des endgültigen Kommandos ist dann also ein zweistufiger Vorgang:

1. Erzeugung der Kommandozeile bis zur Übergabe an die Shell mit der Return-Taste
2. Interpretation der Kommandozeile durch die Shell mit Aliasing, Quoting, Historymechanismus, Variablenauswertung und Dateinamenexpansion

3.2.6.2 Der History–Mechanismus der C–Shell

Mit dem Kommando

`set history=` *n*

kann man dafür sorgen, dass sich die C–Shell die n letzten Kommandos merkt. In der Datei `.cshrc` ist meist ein Wert voreingestellt und kann dort auch verändert werden. Bei dem Aufruf

`history`

werden einem die letzten n Kommandos angezeigt. Die Kommandos sind fortlaufend durchnumeriert. Mit folgenden Aufrufen kann man diese Kommandos wieder ansprechen:

- !! Das letzte Kommando wird nochmal ausgeführt.
- !2 Das Kommando mit der Nummer 2 wird ausgeführt.
- !v Das letzte Kommando, das mit einem 'v' anfängt, wird ausgeführt.

Es können auch Teile der Kommandos angesprochen oder ersetzt und Zusätze gemacht werden.

```
svr50> set history=3
svr50> ls
```

```
clean      test       testdatei  texte
svr50> pwd
/usr/home5/hansen
svr50> history
    44  ls
    45  pwd
    46  history
svr50> !p
pwd
/usr/home5/hansen
svr50> !44 -l clean
ls -l clean
-rwxr-x---  1 hansen          9 Sep 29 17:20 clean
svr50>
```

Oft möchte man gerade die zuletzt ausgeführten Befehle nocheinmal benutzen. Da bietet es sich an, in der Historyliste mit den Pfeiltasten [↓] und [↑] vor- und zurückzugehen.

3.2.6.3 Dateinamenexpansion bei Verwendung von Metazeichen

Eine der nützlichsten Eigenschaften der Shell ist die Expansion von Dateinamen bei Verwendung von Metazeichen. Dateinamen können also reguläre Ausdrücke sein. Die Verwendung von Metazeichen werden im Abschnitt 3.2.9.4 erklärt. Bei der Dateinamenexpansion sind die Möglichkeiten etwas eingeschränkter und wie folgt gegeben.

- ~ wird von der C–Shell durch den Namen des Heimatverzeichnisses ersetzt
- * steht für eine beliebige Zeichenfolge (auch die leere), jedoch nicht für Dateinamen, die mit einem Punkt beginnen
- ? steht für ein beliebiges einzelnes Zeichen
- [...] steht für eines der in Klammern angegebenen Zeichen

Beispiel:

```
svr50> ls -l
total 4
-rwxr-xr--  1 hansen          9 Sep 29 17:20 clean
-rw-r--r--  1 hansen         68 Oct 13 17:50 test
-rw-------  1 hansen          9 Sep 29 15:54 testdatei
drwx------  2 hansen        512 Sep 29 15:38 texte
svr50> ls *a*
clean      testdatei
svr50>
```

Besonders hilfreich ist die automatische Dateinamenexpansion der Shell mittels der Taste [TAB]. Man muss nur den Anfang des Dateinamens tippen und dann [TAB] betätigen. Die Shell vervollständigt den Namen soweit das eindeutig möglich ist.

3.2.6.4 Umlenkungsmechanismen

Unix behandelt Geräte wie Dateien. Alle Gerätedateien sind im Verzeichnis `/dev` registriert.

Normalerweise liest ein Programm von Standard–Input und schreibt auf Standard–Output. Die Fehler werden auf Standard–Error geschrieben. Dabei identifiziert die Shell den Standard–Input mit der Tastatur, den Standard-Output und Standard–Error mit dem Bildschirm. Diese Standardzuweisungen können aber auch umgelenkt werden, so dass von einer Datei gelesen bzw. auf eine Datei geschrieben werden kann. Zur Umlenkung des Standard–Input wird das Zeichen `<` benutzt.

kommando `<` *infile*

Zur Umlenkung des Standard–Output wird das Zeichen `>` benutzt.

kommando `>` *outfile*

In diesem Fall wird eine eventuell schon vorhandene Datei *outfile* überschrieben! Soll der Standard–Output an eine Datei angehängt werden, so benutzt man die Zeichen `>>`.

kommando `>>` *outfile*

In beiden Fällen würden Fehlermeldungen trotzdem auf dem Bildschirm erscheinen. Die C–Shell bietet die Möglichkeit, Standard–Output und Standard–Error in eine gemeinsame Datei umzuleiten.

kommando `>&` *outfile*

Schließlich gibt es noch eine andere Art der Eingabe, die als „here document“ bezeichnet wird.

kommando `<<` *endstring*

Statt von Standard–Input werden jetzt die nachfolgenden Zeilen gelesen, und zwar solange, bis eine Zeile lediglich *endstring* enthält. Beispiel:

```
svr50> cat > test << EOF
Dies ist ein Test.
So koennte man ohne Editor einen Text eintippen.
EOF
svr50> cat test
Dies ist ein Test.
So koennte man ohne Editor einen Text eintippen.
svr50>
```

In der Regel werden solche Konstruktionen in Kommandoprozeduren eingesetzt.

3.2.6.5 Filter

Filter sind Programme, die Text von Standard–Input lesen und irgendwie modifiziert auf Standard–Output schreiben. Wird einem Filter als Parameter ein Dateiname gegeben, so wird statt von Standard–Input von der Datei gelesen. Wichtigste Filter sind `wc`, `sort` und `grep`.

Das Kommando

`wc` [*datei ...*]

zählt die Zeilen, Wörter und Zeichen von Dateien. Das Kommando

`sort` [*datei*]

sortiert die Zeilen von *datei* lexikalisch. Das Kommando

`grep` [`-i`] *regular-expression* [*datei ...*]

sucht in den angegebenen Dateien nach dem regulären Ausdruck *regular–expression* und gibt jede Trefferzeile aus. Mit der Option `-i` wird nicht zwischen Groß- und Kleinschreibung unterschieden.

3.2.6.6 Pipes

Soll in einer Kommandosequenz die Ausgabe des einen Kommandos als Eingabe des nachfolgenden Kommandos benutzt werden, so kann dies durch das Pipe–Symbol | geschehen.

kommando1 | *kommando2* | ...

Pipes werden oft in Kombination mit Filtern benutzt. Beispiel:

```
svr50> ls | sort
clean
test
testdatei
texte
svr50> ls -l | wc
       5      34     215
svr50>
```

3.2.6.7 Drucken

Da Unix ein multi–user, multi–tasking Betriebssystem ist, sollte man normalerweise Dateien nicht direkt zum Drucker schicken, sondern das sog. Druckerspoolsystem benutzen. Dieses verwaltet eine Warteschlange, damit die verschiedenen Dateien nicht gemischt werden. Außerdem kann der Druckerspooler so konfiguriert werden, dass für jeden Druckjob ein Deckblatt mit dem Benutzernamen ausgegeben wird.

Mit dem Kommando

`lpr` [`-P`*drucker*] [*datei ...*]

wird die Datei *datei* auf dem Drucker mit dem Namen *drucker* ausgegeben. Fehlt die Angabe *datei*, so wird von Standard–Input gelesen, siehe Kapitel 3.2.6.4. Informationen über die Warteschlange erhält man mit

`lpq` [`-P`*drucker*]

Druckjobs löschen kann man mit dem Kommando

`lprm` [`-P`*drucker*] *druckjobnummer*

Die Angabe des Druckers mit der Option -P kann in unserem Fall in allen Kommandos entfallen, da wir nur einen Drucker benutzen, der als Default–Drucker in allen Kommandos eingestellt ist. Innerhalb unseres Rechner–Pools erhält man mit

`lpquota`

den aktuellen Stand des verbrauchten Papiers des laufenden Monats.

Hinweis: Dieser Befehl ist nur für das Rechnernetz hier im HFI implementiert.

Das Kommando

`a2ps` [option] [datei]

formatiert Dateien für die Ausgabe auf einem Postscript–Drucker. Ohne Angabe von *datei* wird von der Standardeingabe gelesen. Die Ausgabe erfolgt immer im Postscipt–Format, das Ziel der Ausgabe hängt allerdings von der benutzten Option ab. Im Folgenden stellen wir einige nützliche Optionen vor:

-o *datei* Ausgabe in Datei *datei* schreiben

-P *drucker* Ausgabe an Drucker *drucker* senden

-d Ausgabe an Standarddrucker senden

-r Drucke im *Landscape Mode*

-R Drucke im *Portrait Mode*

–columns *anzahl* Druckt *anzahl* Spalten pro Blatt

–rows *anzahl* Druckt *anzahl* Zeilen pro Blatt

Weitere Optionen findet man auf den Man–Pages (`man a2ps`) zu `a2ps`. Zur Funktionsweise von `man` sei auf Abschnitt 3.2.6.8 verwiesen.

3.2.6.8 Online–Dokumentation

Unixsysteme sind normalerweise mit einer Online–Dokumentation aller Kommandos, den sogenannten Man–pages, ausgestattet. Zum Lesen dieser Dokumentation steht das Kommando `man` zur Verfügung.

`man` *kommandoname*

Insbesondere die Erklärungen zu den verschiedenen Optionen eines Befehls sind wichtig für die effektive Benutzung eines Unix–Systems.
Beispiel:

```
svr50> man pwd

PWD(1L)                                                    PWD(1L)

NAME
       pwd - print name of current/working directory

SYNOPSIS
       pwd
       pwd {--help,--version}

DESCRIPTION
       This  manual  page  documents the GNU version of pwd.  pwd
       prints the fully resolved name of the  current  directory.
       That is, all components of the printed name will be actual
       directory names -- none will be symbolic links.

       Note that most Unix shells provide a built-in pwd  command
       with  similar  functionality so the unadorned, interactive
       pwd command will usually execute the built-in version  and
       not this one.

   OPTIONS
```

```
       --help Print  a  usage message on standard output and exit
              successfully.
line 1
```

Die Anzeige stoppt, wenn das Fenster voll ist. Mit der Space–Taste kann man den nächsten Teil anzeigen lassen.

```
       pwd
       pwd {--help,--version}

DESCRIPTION
       This  manual  page  documents the GNU version of pwd.  pwd
       prints the fully resolved name of the  current  directory.
       That is, all components of the printed name will be actual
       directory names -- none will be symbolic links.

       Note that most Unix shells provide a built-in pwd  command
       with  similar  functionality so the unadorned, interactive
       pwd command will usually execute the built-in version  and
       not this one.

   OPTIONS
       --help Print  a  usage message on standard output and exit
              successfully.

       --version
              Print version information on standard  output  then
              exit successfully.

FSF                    GNU Shell Utilities                     1

line 11/66 (END)
```

Wenn `(END)` erscheint ist die Man–page beendet und man gelangt mit [*q*] wieder zum Prompt.

Man kann die Man–pages auch mit `xman` unter X–Window, s. 3.2.8 auf Seite 55, lesen.

Eine weitere Möglichkeit etwas über Kommandos zu erfahren ist

`apropos` *kommandoname*

Damit werden die Man–pages durchsucht, z.B.

```
svr50> apropos apropos
apropos (1)          - search the manual page names and descriptions
svr50>
svr50> apropos latex
latex (1)            - structured text formatting and typesetting
lyx (1)              - A graphical frontend for LaTeX Cr n (c) ..
slitex (1)           - make LaTeX slides
transfig (1x)        - creates a makefile for portable LaTeX figures
svr50>
```

3.2.7 Vernetzung

Ein wesentlicher Vorteil von Unix ist, dass die Software für die Vernetzung Teil des Betriebssystems ist. Außerdem ist zumindest bei Workstations die entsprechende Hardware auch bei Minimalausstattung Standard.

Das der Software zugrunde liegende Protokoll ist TCP/IP (**T***ranmission* **C***ontrol* **P***rotocol*/**I***nternet* **P***rotocol*). Hardwaremäßig erfolgt die Verkabelung meist über Ethernet (mit Koaxialkabeln). Es gibt jedoch ständig Weiterentwicklungen zu leistungsfähigeren Netzen, z.B. FDDI (**F***iber* **D***istributed* **D***igital* **I***nterface*), ATM (**A***synchronous* **T***ransfer* **M***ode*) oder Fast Ethernet.

Aufbauend auf TCP/IP sind die Rechner der TU Berlin vernetzt. Das Netz der TU–Berlin hat einen Zugang zum weltweiten Netzwerk Internet[2]. Jeder an diesem Netz hängende Rechner besitzt eine eindeutige Internetadresse. Sie besteht aus 32 Bit[3], die mit vier durch Punkte getrennte Zahlen im Bereich 0–255 dargestellt werden. Der in den Beispielen benutzte Rechner svr50 hat z.B. die Internetadresse

130.149.47.230

Der weltweit eindeutige Name von svr50 ist

svr50.pi.TU-Berlin.DE

Bei den meisten Kommandos kann man entweder die Internetadresse oder den Rechnernamen benutzen. Unix arbeitet letztlich immer mit der Internetadresse. Abbildung 3.6 zeigt die Einbindung des Micropools in das TU–Netz und weitergehend das Internet.

Wir werden in den folgenden Abschnitten auf die wichtigsten Dienste etwas näher eingehen.

3.2.7.1 NFS — NIS

Das Netzwerk soll für den Benutzer völlig transparent sein. Das bedeutet, egal von welchem Rechner er sich einloggt, soll er immer das gleiche Passwort benutzen können und die gleiche Umgebung vorfinden. Dazu dienen der **N***etwork* **I***nformation* **S***ervice* — *NIS* und das **N***etwork* **F***ile* **S***ystem* — *NFS*. Per NIS stehen netzweit die gleichen Benutzer- und Gruppenkennungen zur Verfügung und NFS ermöglicht die Verteilung des Dateisystems über das Netzwerk.

3.2.7.2 Einloggen auf entfernten Rechnern

Mit dem Kommando

`telnet` *host*

baut man eine Verbindung zum Rechner *host* auf, um sich dort einzuloggen. Für *host* kann der Rechnername oder die Internetadresse benutzt werden. Die Verbindung wird abgebaut, indem auf dem entfernten Rechner `exit` eingegeben wird. Kann `telnet` keine Verbindung aufbauen, so muss man `telnet` mit `quit` verlassen.

Oftmals stimmt auf dem entfernten Rechner die Terminaleinstellung nicht. Im allgemeinen hilft dann

```
set term = vt100
```

[2] interessante Informationen zum Aufbau und der Entwicklung finden sich im WWW (s. 3.2.7.3) unter: http://www.Austria.EU.net/iic/de/index.html

[3] z.Zt. gibt es Entwicklungen zu einem neuen Standard (IPv6 bzw. IPng), in dem Internetadressen dann 128 Bit lang sein werden

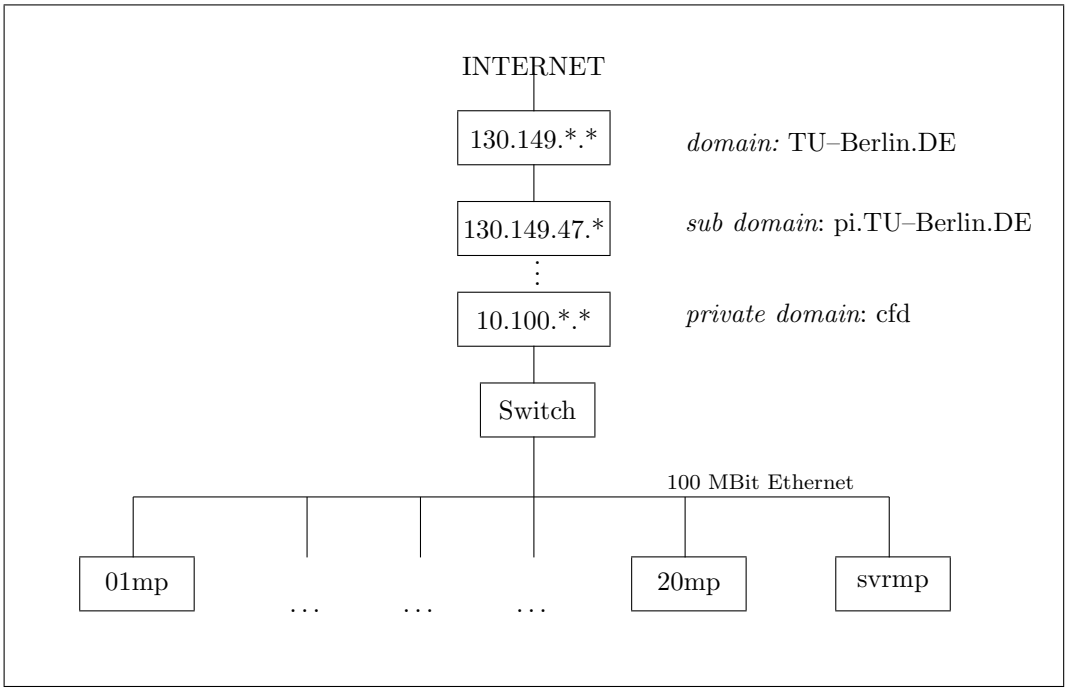


Abbildung 3.6: Netzanbindung des Rechnerpools

3.2.7.3 World Wide Web — WWW

Das Internet hat wohl nur durch die rasante Entwicklung des WWW in den letzten Jahren so eine Verbreitung gefunden. Was also ist das WWW? Das World Wide Web ist *nicht* das Internet, wie so oft dargestellt, sondern nur ein weiterer Dienst. Das WWW ist eine Vielzahl von Hypertext–Dokumenten, die über Links verknüpft sind. Während andere Dienste meistens textorientiert sind, ist das WWW von vornherein als grafischer Dienst konzipiert. Die Daten werden als Seiten mit Grafiken, etc. dargestellt und mit Links zu anderen Seiten irgendwo im Internet verbunden. Außerdem besteht die Möglichkeit Sounds einzubinden. Für die Darstellung ist ein Browser[4] notwendig. Wir haben auf unseren Linux–PCs den Netscape Navigator installiert.

Die Seiten werden in HTML geschrieben. HTML — **H**yper**T**ext **M**arkup **L**anguage ist eine standardisierte Seitenbeschreibungssprache für das WWW. Einführungen in HTML gibt es zahlreich im Netz. Auf der Web–Seite[5] zur Veranstaltung EDV 1 stehen entsprechende Links.

Wie und wozu kann man das WWW nun verwenden? Zuerst einmal wird eine Seite über eine URL — **U**niform **R**essource **L**ocator — aufgerufen. Allgemein setzt sich eine URL folgendendermaßen zusammen: `service://address/[directory:port/document]`.
Gültige Angaben für den Dienst können der Tabelle 3.2.7.3 entnommen werden.

Wenn man sich eine Web–Seite anschaut, erkennt man, dass bestimmte Textstellen mit einer anderen Farbe (meistens blau) und unterstrichen dargestellt sind. Bewegt man die Maus auf solch eine Textstelle, verändert sich der Mauszeiger und in der Statuszeile des Browsers wird (meist) eine URL angezeigt. Diese Textstelle ist also ein Hyperlink und ein Mausklick darauf bringt einen zur der entsprechenden Seite. Auch Bilder können Hyperlinks sein. Diese sind dann (meist) blau umrandet. Damit kann man sich dann durch das „gesamte“ WWW bewegen.

[4] Die bekanntesten sind wohl der MS Internet Explorer und der Netscape Navigator.
[5] http://hodgson.pi.tu-berlin.de/Vorlesungen/EDV1

Dienst	Erklärung	Beispiel
http://	**H**yper**t**ext **T**ransfer **P**rotocol Übertragungsprotokoll für HTML–Dokumente	http://hodgson.pi.tu-berlin.de
ftp://	**F**ile **T**ransfer **P**rotocol — s 3.2.7.4	ftp://ftp.fu-berlin.de
mailto:	Schreiben einer E–Mail — s. 3.2.7.5	mailto:hansen@cfdmail.pi.tu-berlin.de
news:	Lesen von News	news:bln.lehre.misc
telnet:	Starten einer Telnet Sitzung — s.3.2.7.2	telnet:*host*

Tabelle 3.1: Dienste des Internet

Um alle angegebenen Dienste mit Hilfe des Browsers nutzen zu können, müssen einige Einstellungen vorgenommen werden. Im bei uns installierten Browser „Netscape“ gibt es in der Menüleiste den Eintrag `Options` und darunter `Mail and News Preferences`. Nach der Auswahl erscheint ein Fenster mit mehreren Unterabschnitten. Viele Felder davon sind schon ausgefüllt, nur die unter `Identity` müssen von jedem Benutzer einzeln noch ausgefüllt werden.

3.2.7.4 File Transfer Protocol — FTP

FTP ist ein weiterer Dienst des Internets. Damit ist es möglich, Dateien von einem Computer auf einen anderen zu übertragen. Es gibt verschiedene Möglichkeiten, diesen Dienst zu nutzen. In jedem Fall ist ein Client notwendig. Es gibt kommandozeilen- oder fensterorientierte ftp–Clients. So ist z.B. in den meisten Browsern heutzutage ein grafisch aufbereiteter ftp–Client enthalten. Wir wollen hier dennoch eine kurze Einführung in das ftp–Programm unserer Linux–PCs geben. Der Client wird mit dem Kommando

`ftp` *[host]*

gestartet. Man erhält dann den ftp–Prompt: `ftp>`. Mit `?` erhält man die Liste der möglichen Kommandos. Mit der Angabe *host* beim Start wird direkt eine Verbindung zu dem angegebenen Rechner hergestellt. Hat man das Programm ohne *host* gestartet muss man mit

`open` *rechnername*

eine Verbindung zu einem Rechner aufbauen. Grundsätzlich kann zu jedem Internetrechner der Welt eine Verbindung hergestellt werden. Man muss allerdings die Zugangsrechte über eine Benutzerkennung und ein Passwort haben, genau wie beim lokalen Zugang zu unseren Linuxrechnern. Es gibt viele ftp–Server, die einen anonymen Zugang erlauben. Dort gibt man bei der Frage nach dem Name und dem Passwort `anonymous` ein. Manchmal wird auch abweichend davon `guest` oder `ftp` erlaubt. Nach dem erfolgreichen Einloggen bei einem ftp–Server ist es möglich, sich mit

`cd` *verzeichnis*

siehe Seite 34, durch den Dateibaum zu bewegen. Zum Herunterladen von Dateien gibt es den Befehl

`get` *dateiname*

oder

`mget` *dateiname*

Möchte man Dateien zu einem Rechner hinaufladen, nutzt man

`put` *dateiname*

oder

`mput` *dateiname*

Die Benutzung von `mget` und `mput` erlaubt die Nutzung von Metazeichen zur Angabe der zu übertragenden

Dateien.

Dabei ist allerdings zu beachten, dass es zwei verschiedene Transfermodi gibt. Für reine Textdateien ist *ascii*, für alle anderen *binary* der richtige Transfermodus. Man kann mit `asc` bzw. `bin` die Übertragungsart einstellen.

Mit

`close`

beendet man die Verbindung zu einem Rechner. Will man den ftp–Client komplett beenden kann man auch direkt

`quit`

eintippen. Das Kommando

`help` *befehl*

am ftp–Prompt gibt weitere Hinweise aus.

Öffentliche ftp–Server, wie z.B. ftp.fu-berlin.de, können auch mit dem in den Netscape integrierten ftp–Client genutzt werden, um Daten zu übertragen. Meist ist es aber nur möglich Dateien herunter- und nicht hinaufzuladen.

3.2.7.5 Elektronische Post — E–Mail

Der weltweit wohl am häufigsten genutzte Internetdienst ist das elektronische Versenden von Briefen — E–Mail. Das funktioniert im wesentlichen wie der Name vermuten lässt: Man schreibt einen Text, versieht ihn mit Namen und Adresse des Empfängers und schickt ihn ab. Wie sieht nun eine solche Adresse aus? Der „Name“ des Empfängers ist die Benutzerkennung seines Accounts und die Adresse ist der Domain–Name. Mit dem Account auf unserem Netz hat jeder Nutzer auch eine E–Mail–Adresse. Der Username und der Domain–Name werden durch ein @ zur vollständigen Anschrift zusammengesetzt und würde für unseren Nutzer `hansen` folgendermaßen lauten:

hansen@cfdmail.pi.tu-berlin.de

Es gibt viele verschiedene Möglichkeiten, Mails zu verschicken und zu empfangen.

3.2.8 X–Window — eine grafische Benutzeroberfläche

3.2.8.1 Einführung

Das X–Window System ist ein leistungsfähiges und portables Fenstersystem für Grafikbildschirme, das am Massachusetts Institute of Technology (MIT) entwickelt wurde. Das X–Window System (im folgenden einfach X genannt) läuft auf zahlreichen Rechnersystemen. Es ist unabhängig von Betriebssystem, Prozessor, Hersteller und Programmiersprache. Allerdings hat es die größte Verbreitung auf Unix–Workstations und hat sich dort zum Standard entwickelt. X hat folgende wesentliche Vorteile:

- X ist netzwerktransparent. Anwendungsprogramme können gleichzeitig und ebenso einfach auf Bildschirmen anderer Rechner irgendwo im Netzwerk eine Ausgabe machen wie auf ihrem eigenen Bildschirm.
- Das benutzte Kommunikationsverfahren ist sprachen- und rechnerunabhängig. Die anderen Rechner müssen weder vom selben Typ sein noch unter demselben Betriebssystem laufen.

- X ist nicht Teil des Betriebssystems. Für das Betriebssystem ist es lediglich ein weiteres Anwendungsprogramm.
- X ist für jederman zu den Vertriebskosten frei verfügbar.

3.2.8.2 Server, Client und Kommunikationskanal

Im Gegensatz zu den meisten anderen Fenstersystemen ist X kein homogenes Einzelprogramm. Es besteht vielmehr aus drei zusammenhängenden Teilen:

- Der Server ist ein Programm, das das physikalische Display und die Eingabegeräte steuert.
- Die Clients sind Programme, die vom Server die Ausführung spezieller Fensteroperationen anfordern.
- Der Kommunikationskanal wird von Server und Clients benutzt, um untereinander Verbindung zu halten.

Der Server
Die erste Grundkomponente von X ist der Server, die Software, die den physikalischen Bildschirm und die Eingabegeräte Tastatur und Maus kontrolliert. Der Server erzeugt auf Anforderung eines Client–Programms die Fenster und zeichnet Bilder und Texte in sie hinein. Von sich aus macht der Server nichts, er führt lediglich Aktionen aus, die von Client-Programmen angefordert wurden.

Jeder Bildschirm hat genau einen Server.

Der Server wird vom Hersteller des X–Systems geliefert und normalerweise vom Benutzer nicht geändert. Für das Betriebssystem ist der Server jedoch nur ein weiteres Benutzerprogramm.

Client–Programme
Die zweite Grundkomponente von X bilden die Client–Programme. Das sind Anwendungsprogramme, die von den Fensterfähigkeiten des Systems Gebrauch machen. Anwendungsprogramme unter X werden Clients genannt, da sie tatsächlich Kunden des Servers sind. Sie bitten den Server, bestimmte Aktionen für sie auszuführen. Sie können kein Fenster und keinen Bildschirm direkt beeinflussen. Sie können lediglich den Server bitten, das zu tun, was sie wünschen. Typische Anfragen wären z.B.: „Schreibe die Zeichenkette 'Hallo, Welt' in das Fenster XYZ“ oder „Zeichne eine Linie einer bestimmten Farbe im Fenster ABC von D nach E“.

Selbstverständlich bildet das Anfordern von Fensteroperationen nur einen Teil eines Client–Programms. Der andere Teil ist der Programmcode, der sich auf die Aufgabe bezieht, die der Benutzer ausführen will. Als Benutzer verwendet man meist Client–Programme aus verschiedenen Quellen. Einige wesentliche werden zusammen mit dem System geliefert, andere erhält man von Fremdanbietern und manche Benutzer schreiben Ihre eigenen Client–Programme für bestimmte Anwendungen.

Der Kommunikationskanal
Die dritte Grundkomponente von X ist der Kommunikationskanal. Die Client–Programme nutzen ihn, um Anforderungen an den Server zu schicken, und der Server sendet über ihn Status- und andere Informationen an die Client–Programme zurück.

Die genaue Beschaffenheit des Kommunikationskanals ist nicht so wichtig, solange sowohl Server als auch Client wissen, wie er zu benutzen ist. Die auf einem bestimmten System oder Netzwerk benötigten Kommunikationstypen werden durch die X–Window Bibliothek (engl. *library*) für dieses System unterstützt. Die gesamte Kommunikation zwischen Client und Server wird über diese Bibliothek, die „Xlib“ heisst, abgewickelt. Als Folge davon können alle Client–Programme automatisch sämtliche Kommunikationsmethoden benutzen, indem sie auf die Bibliothek zurückgreifen.

Die Client/Server–Kommunikation gliedert sich in zwei große Klassen, die die beiden wesentlichen Operationsweisen im X–System widerspiegeln:

- Server und Client laufen auf demselben Computer. Hier können sie jede auf diesem Rechner verfügbare Form der Kommunikation zwischen Prozessen (IPC=engl. *inter–process–communication*) nutzen. In diesem Modus verhält sich X letztendlich wie zahlreiche konventionelle Fenstersysteme.
- Server und Client laufen auf verschiedenen Computern. In diesem Fall müssen sie über ein Netzwerk kommunizieren, indem sie ein gegenseitig vereinbartes Protokoll benutzen.

Die Fähigkeit einer Anwendung, genauso problemlos innerhalb eines Netzwerks wie auf lokalen Rechnern zu funktionieren, wird Netzwerktransparenz genannt und findet sich fast nur bei X. Durch diese Besonderheit ist X für den Aufbau von anpassungsfähigen Vielzweck–Netzwerken mit zusammenarbeitenden Rechnern geeignet.

Da Server und Client völlig getrennt sind, hat sich ein neuer Terminaltyp entwickelt, der X–Terminal genannt wird. Ein X–Terminal ist eine abgespeckte Workstation, auf der nur der X–Server laufen soll und sonst nichts. Diese verfügt über eine Tastatur, eine Maus, einen Bildschirm und kann im Netz kommunizieren. Sie verfügt jedoch nicht über ein eigenes Dateisystem und unterstützt keine Mehrzweckprogramme. Diese müssen irgendwo anders im Netzwerk laufen.

3.2.8.3 Windowmanager

Das X–Grundsystem, also der Server, bietet nur die elementaren Fensterfunktionen:

- Fenster erzeugen
- In Fenster schreiben und zeichnen
- Tastatur- und Mauseingaben verarbeiten
- Fenster entfernen

Die komfortable Manipulation der Fenster wird bei X von einem separaten Programm wahrgenommen, dem sogenannten Windowmanager. Er bietet mindestens folgende Funktionen, die alle mit der Maus gesteuert werden:

- Verschiedene Anwendungsfenster erzeugen
- Bewegen der Fenster auf dem Bildschirm
- Ändern der Fenstergröße
- Verdeckte Fenster nach oben holen und umgekehrt
- Anwendungen starten und beenden
- Bildschirm neu zeichnen
- Fenster in Icons und Icons wieder in Fenster umwandeln

Ähnlich wie bei Unix verschiedene Shells zur Auswahl stehen, so gibt es auch viele verschiedene Windowmanager. Der Windowmanager, der vom MIT im sogenannten Core–Release mitgeliefert wird, ist der `uwm`. Jeder Nutzer hat die Möglichkeit seinen bevorzugten Windowmanager zu benutzen, sofern dieser installiert ist.

3.2.8.4 Benutzung von X

Um X benutzen zu können, muss man mindestens den Server starten. Oftmals (speziell bei unseren Linux–PCs) sind Unix–Systeme so konfiguriert, das X immer läuft. Startet man jedoch im einfachen Textterminalmodus, so muss der einzelne Anwender X starten. Dazu dient das Kommando

```
xinit
```

das den X–Server startet und danach ein `xterm` Fenster öffnet. In diesem `xterm` Fenster, das sich wie ein normales Textterminal verhält, kann man mit dem Kommando

```
uwm &
```

z.B. den Standardwindowmanager starten. Der Windowmanager wird das Starten von weiteren Textterminalfenstern ermöglichen. Man kann aber auch über die Tastatur das Kommando

```
xterm &
```

eingeben. Oftmals werden der Befehl `xinit` und der Aufruf des Windowmanager in einem Kommando zusammengefasst, z.B.

```
startx
```

Damit Clients von entfernten Rechnern Fenster auf dem Bildschirm öffnen können, muss diesen mit dem Kommando

`xhost +` [*rechnername*]

Zugriffsrecht gewährt werden. Fehlt die Angabe *rechnername*, so findet keine Zugangskontrolle mehr statt. Zugriffsrechte werden entzogen mit dem Kommando

`xhost -` [*rechnername*]

Man beachte, dass das X–System Zugriffsrechte an Rechner verteilt, und nicht an bestimmte Benutzer! Um auf einem entfernten Rechner einen Client zu starten, der ein Fenster auf dem lokalen Bildschirm öffnen soll, muss auf dem entfernten Rechner die Variable 'DISPLAY' entsprechend gesetzt werden.

`setenv DISPLAY` *rechnername*`:` *displaynummer*

rechnername ist der Name des Computers, auf dem der X–Server läuft. Der normale Bildschirm hat die *displaynummer* 0.
Wenn jemand mal für kurze Zeit den Rechner verlassen muss, sollte er ihn nicht einfach so stehen lassen, da somit jede Person Zugriff auf den Homebereich hat und dort Dateien z.B. löschen könnte. Es gibt das Kommando

```
xlock
```

Damit wird der Rechner gesperrt, und nur mit dem Passwort kann wieder darauf zugegriffen werden. Man sollte allerdings einen Rechner nicht zu lange blockieren, da ja auch andere Nutzer arbeiten möchten. Deshalb ist in unserem PC–Pool der `xlock` so konfiguriert, dass nach 5 Minuten jeder einen 'gelockten' Rechner zugänglich machen kann, indem er den Benutzer ausloggt. Damit gehen auch alle nichtgespeicherten Daten verloren.

Zum Ende dieses Abschnitts noch ein Hinweis für seltene **Notfälle:** Sollte der Windowmanager im Verlauf einer Sitzung gar nicht mehr reagieren, kann man mit der Tastenkombination [*Control*][*Alt*][*Backspace*] den X–Server „abschießen“. Dabei gehen aber alle nicht gespeicherten Informationen verloren und man wird ausgeloggt.

3.2.8.5 Weitere Anwendungsprogramme unter X

Wir wollen hier nur kurz noch einige Anwendungen erwähnen, die auf den Rechnern des Hermann–Föttinger–Insituts installiert sind. Für die **Textbearbeitung** gibt es den `xemacs`, ein Editor mit Menüs und Buttons zur Benutzung mit Mausunterstützung. Zum Betrachten von **Postscript–Dateien**[6] gibt es den `ghostview`. Mit LaTeX erstellte dvi–files können mit dem `xdvi` dargestellt werden.

Eine Alternative zu `ghostview` stellt das Programm `gv` dar, dass ebenfalls als User Interface zu `ghostscript` fungiert. Es bietet eine Reihe von Optionen zum Anzeigen und Drucken von PS–Dateien.

Das Programm `acroread` der Firma Adobe dient dem Betrachten von Dateien, die als PDF (**P**ortable **D**ocument **F**ormat) – Dokumente vorliegen. Es ist kostenlos für eine Reihe von Plattformen erhältlich, u.a. auch für Linux und Windows.

`Acroread` wie auch `gv` dienen lediglich dem Betrachten von Dokumenten, eine Bearbeitung der Dateien ist damit nicht möglich.

Zur Betrachtung und Bearbeitung von Bildern verschiedenster Formate gibt es `xv` (Xview).

3.2.9 Editor — vi

Unix stellt verschiedene Editoren zur Verfügung, von denen jeder spezielle Vor- und Nachteile hat. Wir werden hier den `vi` näher vorstellen, da er auf allen Unixsystemen vorhanden ist. vi arbeitet immer mit einer lokalen Kopie der Datei im Arbeitsspeicher. Solange nicht abgespeichert wird, bleibt also die Datei auf der Festplatte unverändert.

3.2.9.1 Aufruf und Verlassen des Editors

Der Editor wird aufgerufen mit

`vi` [`-r` *recoverdatei*] [`-x`] [`-R`] [*datei ...*]

Die Option `-r` *recoverdatei* wird benutzt, wenn während des Editierens der Editor oder das ganze System abgestürzt sind. Durch die Option `-x` erfolgt eine Verschlüsselung der Daten. Der Benutzer wird dabei interaktiv nach einem Schlüssel gefragt. Für diesen erscheint bei der Eingabe kein Echo. Durch die Option `-R` (**r**ead only) wird die Datei nur zum Lesen geöffnet. Gibt man eine bestehende *datei* an, wird diese geöffnet, ansonsten wird eine neue Datei mit dem angegebenen Namen erstellt.

Der Editor wird mit dem Kommando

`ZZ`

verlassen. Wurde die Datei verändert, so wird vorher abgespeichert. Mit dem Kommando

`:w` [*datei*]

wird nur abgespeichert. Ohne Angabe von *datei* wird der beim Aufruf angegebene Dateiname benutzt. Den Editor zu verlassen, ohne Änderungen abzuspeichern, ist mit dem Kommando

`:q!`

möglich. Wurde beim Aufruf des `vi` mehr als eine Datei angegeben, so wechselt man mit

`:n`

zur nächsten Datei, und mit

[6] Postscript ist eine weitverbreitete Seitenbeschreibungssprache

`:rew`

kommt man wieder in die erste Datei.

3.2.9.2 Cursorpositionierung

Zuerst die schlechte Nachricht: Cursorpositionierung per Maus ist nicht möglich. Die gute Nachricht: Alles per Tastatur denkbare ist möglich. Es gibt dabei vier Arten der Verschiebung:

- In festen Bildschirmeinheiten wie Zeichen, Zeile, halbe Ausschnittsgröße oder um eine ganze Ausschnittsgröße (Seite)
- In Objekten wie Wort, Satz oder Absatz
- Durch Suchen nach einem Textmuster, siehe Abschnitt 3.2.9.4.
- Korrespondierende Klammerpaare

Die Objekte Wort, Satz und Absatz sind wie folgt definiert:

- Ein **Wort** ist eine Folge von Buchstaben und Ziffern ohne Zwischenraum. Wird bei den Wort–Operationen der kleine Buchstabe verwendet, so wird ein Sonderzeichen als eigenes Wort interpretiert. Bei den großen Buchstaben werden Worte nur durch Leerzeichen, Tabulator oder Neue–Zeile getrennt.
- Ein **Satz** ist eine Folge von Worten, die durch '.', '!' oder '?' terminiert wird. Diesem Zeichen müssen entweder eine neue Zeile oder zwei Leerzeichen folgen.
- Ein **Absatz** beginnt nach jeder Leerzeile.

Die Objekte sind nicht nur zur Cursorpositionierung wichtig, sondern auch beim Löschen und Kopieren etc. Die Objekte (`<obj>`) werden ausgewählt durch

`< >` (Leerzeichen) für ein einzelnes Zeichen,
`w` für ein Wort ohne Sonderzeichen,
`W` für ein Wort mit Sonderzeichen,
`b` für ein vorhergehendes Wort ohne Sonderzeichen,
`B` für ein vorhergehendes Wort mit Sonderzeichen,
`G` für bis zum Ende des Puffers,
`^` für bis Anfang der Zeile,
`$` für bis zum Ende der Zeile.

Einfache Cursorbewegungen

`h`	1 Zeichen nach links
`j`	1 Zeichen nach oben
`k`	1 Zeichen nach unten
`l`	1 Zeichen nach rechts
`<cr>`	Erste Zeichen der nächsten Zeile

`-` Erste Zeichen der vorherigen Zeile

`G` Erste Zeichen der letzten Zeile

Dabei sind Wiederholungsfaktoren möglich. Die Eingabe von `5l` würde z.B. den Cursor 5 Stellen nach rechts verschieben. Beim `G` wird die Zeile mit der entsprechenden Zeilennummer angesprungen.

Cursorbewegungen auf der aktuellen Zeile

`0` Anfang der aktuellen Zeile

`$` Letzte Zeichen der aktuellen Zeile

`n|` Bewegt den Cursor auf die n-te Spalte der aktuellen Zeile Dies ist sehr nützlich für Fortranprogrammierer.

Cursorbewegungen auf Worte

`w` Anfang des nächsten Wortes

`W` Anfang des nächsten Wortes

`b` Anfang des aktuellen oder vorangehenden Wortes

`B` Anfang des aktuellen oder vorangehenden Wortes

`e` Ende des aktuellen oder folgenden Wortes

`E` Ende des aktuellen oder folgenden Wortes

Wiederholungsfaktoren sind möglich.

Cursorbewegungen auf Sätze und Absätze

`)` Anfang des nächsten Satzes

`(` Anfang des vorhergehenden Satzes

} Anfang des nächsten Absatzes

{ Anfang des vorhergehenden Absatzes

Cursorbewegungen Bildschirmweise

`H` Erste Zeichen der ersten Zeile des aktuellen Bildschirms

`M` Erste Zeichen der mittleren Zeile des aktuellen Bildschirms

`L` Erste Zeichen der letzten Zeile des aktuellen Bildschirms

`<ctrl+f>` Ein Bildschirm vorwärts

`<ctrl+b>` Ein Bildschirm rückwärts

Positionieren des Cursors auf Klammerpaare

`%` Bewegt den Cursor auf die korrespondierende Klammer (,),{,},[,]

3.2.9.3 Einfügen, Verändern und Löschen

Um etwas einfügen zu können, muss vom Kommandomodus in den Einfügemodus umgeschaltet werden. Der Einfügemodus wird mit `<esc>` verlassen. Man beachte, dass Cursorpositionierungen etc. nur im Kommandomodus möglich sind.

Einfüge- und Ersetzungskommandos

`i`	Einfügen vor dem Cursor
`I`	Einfügen am Zeilenanfang
`a`	Einfügen nach dem Cursor
`A`	Einfügen am Zeilenende
`o`	Einfügen von Text nach der aktuellen Zeile
`O`	Einfügen von Text vor der aktuellen Zeile
`r`	Ersetzen eines Zeichens durch ein anderes
`R`	Überschreiben des Textes
`c<obj>`	Ersetzen des Objektes `<obj>`
`C`	Ersetzen des Rests der Zeile
`s`	Ersetzen des Zeichens durch den eingegebenen Text. Wiederholungsfaktoren sind möglich
`S`	Ersetzen der ganzen Zeile

Im Eingabemodus haben u.a. folgende Kontrollzeichen eine Sonderbedeutung:

Kontrollzeichen im Einfügemodus

`<ctrl+w>`	Lösche das letzte eingegebene Wort
`<bs>`	Lösche das letzte eingegebene Zeichen
`<ctrl+v>`	Eingabe eines Sonderzeichens

Mit

`:r` *datei*

wird die Datei *datei* nach der aktuellen Zeile eingefügt.

Löschen von Text

`x`	Löscht das Zeichen an der Cursorposition
`X`	Löscht das Zeichen vor der Cursorposition
`d<obj>`	Löscht das Objekt `<obj>`
`dd`	Löscht die aktuelle Zeile
`D`	Löscht die Zeile von der aktuellen Cursorposition bis zum Zeilenende
`dG`	Löscht die aktuelle und alle nachfolgenden Zeilen

3.2.9.4 Suchen und Ersetzen, reguläre Ausdrücke

Reguläre Ausdrücke

Häufig möchte man Unix–Kommandos wie z.B. Löschen oder Kopieren auf eine Reihe von Dateien anwenden oder in Texten, statt nach einer festen Zeichenkette, mit einem Muster suchen, in dem noch gewisse Freiheitsgrade vorhanden sind. Für beide Anforderungen stehen unter Unix sogenannte „reguläre Ausdrücke“ (regular expression) zur Verfügung, die weitgehend einheitlich von den verschiedenen Unix-Programmen interpretiert werden.

Ein regulärer Ausdruck ist dabei eine Folge von normalen Zeichen und Metazeichen. Ein normales Zeichen steht für das entsprechende ASCII–Zeichen selbst. Ein Metazeichen ist ein Zeichen, welches nicht das entsprechende Zeichen darstellt, sondern eine erweiterte Bedeutung besitzt. Das Fragezeichen ? z.B. steht in der Angabe von Dateinamen für ein beliebiges (auch nicht druckbares) Zeichen. Meint man in einem regulären Ausdruck das Zeichen selbst und nicht die Metabedeutung, so muss man das Metazeichen maskieren. Dies geschieht durch das Voranstellen des Fluchtzeichens \. Der Editor vi stellt folgende Metazeichen zur Verfügung.

Suchmuster	Bedeutung
`.`	beliebiges einzelnes Zeichen
`.*`	beliebige Zeichenkette (auch die leere)
`*`	beliebige Wiederholung des vorangestellten Zeichens
`[...]`	eines der Zeichen aus ...
`[a-e]`	eines der Zeichen von a bis e
`[a-eh-x]`	eines der Zeichen von a bis e oder h bis x
`[^...]`	ein beliebiges Zeichen außer ...
`\`	Fluchtsymbol
`^muster`	Zeichenkette muster am Anfang der Zeile
`muster$`	Zeichenkette muster am Ende der Zeile
`^muster$`	Zeile, die nur aus der Zeichenkette muster besteht
`\<muster`	Zeichenkette muster am Anfang eines Wortes
`muster\>`	Zeichenkette muster am Ende eines Wortes

Kommandos zum Suchen

`/<reg-exp>`	Suche von der aktuellen Cursorposition aus vorwärts nach dem Ausdruck `<reg-exp>`.
`?<reg-exp>`	Suche von der aktuellen Cursorposition aus vorwärts nach dem Ausdruck `<reg-exp>`.
`n`	Wiederhole das letzte Suchkommando.
`N`	Wiederhole das letzte Suchkommando in umgekehrter Richtung.
`:set ic`	Unterscheide nicht zwischen Groß- und Kleinschreibung.
`:set noic`	Unterscheide zwischen Groß- und Kleinschreibung.

Suchen und Ersetzen

Sollen mehrere gleiche Textteile gesucht und ersetzt werden, so geschieht dies am besten mit dem Substitute–Kommando des ex–Modus.

`:`[*bereich*]`s/`*suchmuster*`/`*ersetzungsmuster*`/`[`gc`]

Die Angabe *bereich* wählt dabei die Zeilen aus, die bearbeitet werden sollen:

bereich	Bedeutung
weggelassen	nur die aktuelle Zeile
`%`	der gesamte Text
`a,e`	Zeile a bis e
`a,$`	Zeile a bis zum Ende der Datei
`.,e`	von der aktuellen Zeile bis Zeile e
`e`	nur die Zeile e

Die Option `g` (global) erlaubt mehr als eine Ersetzung pro Zeile. Durch die Option `c` (confirmation) wird bei jeder Ersetzung nachgefragt.

3.3 Programmübersetzung

3.3.1 Compiler

Compiler **übersetzen** den Quellcode von Programmen, die in Assembler oder in einer höheren Programmiersprache geschrieben sind, in den vom Rechner zu verarbeitenden Objekt- oder Maschinencode (auch Zielcode). Dabei arbeiten verschiedene Komponenten zusammen. Abbildung 3.7 zeigt schematisch, wie auf einem Rechner aus den Source–Files ausführbarer Code generiert wird.

Compiler übersetzen ein Quellprogramm als Ganzes und generieren daraus ein Objektprogramm. Es kann, wenn es nicht verändert werden soll, gespeichert und ohne nochmaliges Übersetzen beliebig oft abgearbeitet werden.

Die Übersetzung läuft in der Regel in **mehreren Durchgängen** („pass“) ab. Der erste Durchgang beinhaltet eine lexikalische Analyse auf der Ebene einzelner Wörter und eine Syntaxüberprüfung des Programms. Dabei werden die Anweisungen und die syntaktische Programmstruktur auf **Zulässigkeit** geprüft. Die Compiler geben Fehlermeldungen aus, die von einfachen Fehlercodes („error no. xxx in line yyy“) bis hin zu komfortablen Diagnosemeldungen mit Angabe der fehlerhaften Programmabschnitte reichen. In weiteren Durchgängen wird eine Tabelle für die Zuordnung von Namen zu Adressen („cross reference list“) erzeugt und das Programm in **Objektcode** übertragen. Hieran können sich Optimierungsläufe anschließen, in denen der Objektcode nach unterschiedlichen Gesichtspunkten (meist Programmausführungszeit — „execution time“) und in verschiedenen Stufen optimiert wird.

Die Compilation lässt sich durch Optionen beim Compileraufruf oder durch spezielle Quellcodeanweisungen steuern. So kann der Ausdruck des Quellcodes, des Übersetzungsprotokolls („compiler listing“) der Fehlermeldungen, des Objektcodes, des Assemblercodes u.a. angefordert werden. Weitere Optionen sind das Einbinden von Quellprogrammen oder -programmteilen aus Bibliotheken in den Quelltext („include“) sowie bedingte Compilation.

Neben den Compilern gibt es die **Interpreter**. Sie gehen befehlsweise vor, indem sie jeweils einen Programmbefehl übersetzen und diesen unmittelbar abarbeiten. Interpretierendes übersetzen, wie es insbesondere von der Programmiersprache BASIC her bekannt ist, hat den Nachteil, dass das Programm bei jeder Ausführung erneut übersetzt werden muss und nicht optimiert werden kann. Von Vorteil ist jedoch die Möglichkeit des interaktiven und befehlsweisen Abarbeitens der Programme. Die Shell ist ein Interpreter, der die eingegebenen Befehle direkt übersetzt und ausführt.

3.3.2 Linker und Lader

Objektmodule, wie sie z.B. der Compiler erzeugt, stellen Folgen von Maschinencodebefehlen dar. Alle Referenzen sind relativ zum Modulanfang adressiert und somit noch nicht an feste Speicheradressen

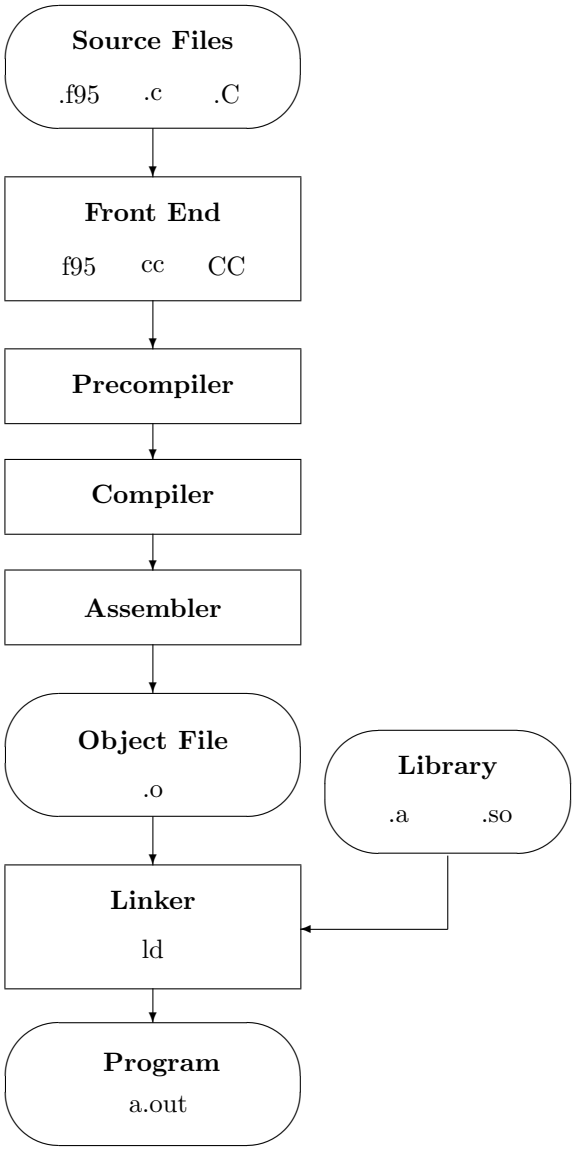


Abbildung 3.7: Überführung des Quellcodes in ein ausführbares Programm

gebunden (**verschiebbar** — „relocatable“). Große Programme bestehen im allgemeinen aus mehreren Objektmodulen. Sie kommen aus getrennter Übersetzung der Programmbausteine (Hauptprogramm, Unterprogramme) oder aus bestehenden Objektmodul–Bibliotheken („libraries“).

Der **Linker** verbindet nun mehrere Objektmodule zu einem lauffähigen Programm (Lademodul). Dabei werden die Adressen dieser Module so modifiziert, dass ein einheitliches Gebilde entsteht. Gleichzeitig stellt der Linker die logischen Bezüge zwischen den einzelnen Objektmodulen her, indem er Referenzen auflöst, die auf Adressen in anderen Modulen verweisen. Solche externen Referenzen bleiben bei der Übersetzung zunächst offen und werden in den Symboltabellen gekennzeichnet. Kann der Linker diese Bezüge nicht befriedigen (d.h. er findet die gerufenen Unterprogramme nicht), erscheint eine Fehlermeldung (z.B. „unsatisfied external reference ...“). Das Lademodul enthält den verschiebbaren Code („relocatable code“) des Lademoduls sowie eine Ladeliste („load map“) mit Angaben über die Positionen der Objektmodule innerhalb des Lademoduls. Bei anderen Lösungen wird eine Liste mit allen Symbolen, die relativ zum Anfang des Lademoduls adressiert sind, aufgebaut.

Der **Lader** („loader“) schließlich ordnet dem Lademodul absolute Adressen im Hauptspeicher zu und transportiert es dorthin.

3.3.3 Die Compiler f95 / gcc

Wir benutzen den Fortran–Compiler f95 und den C–Compiler gcc. Der Aufruf

`f95` [options] *source.f95*

`gcc` [options] *source.c*

startet den Compiler mit dem angegebenen Quellcode. Folgende Optionen sind für den Anfang wichtig:

Option	Bedeutung
-c	übersetzen des Sourcefiles, aber nicht linken es wird ein Objectfile `source.o` erzeugt
-g	debugging Informationen werden erzeugt
-o *file*	Ausgabe wird in den angegebenen Dateinamen geschrieben ohne diese Option wird `a.out` als Ausgabefile erzeugt

3.3.4 Werkzeuge zur Programmerstellung — make

Make ist ein Programm, das Abhängigkeiten zwischen Dateien verwaltet. Oft wird es zur Programmerstellung in Softwareprojekten benutzt, ist aber nicht auf dieses Einsatzgebiet beschränkt.

Make benutzt für die Ausführung eine Datei, in der Datei–Abhängigkeiten, sowie Ziele und die zur Erreichung dieser Ziele notwendigen Aktionen beschrieben werden. Diese Datei heißt standardmäßig *makefile* bzw. *Makefile*, kann jedoch auch einen anderen Namen haben (s. Abschnitt 3.3.4.1).

Ein Ziel ist das Ergebnis einer Aktion.

Eine Aktion ist die Ausführung eines Programmes.

Eine Abhängigkeit ist eine Datei, die für die Erstellung eines Zieles notwendig ist.

Der Aufruf von make sieht wie folgt aus:

`make` [Optionen] [Ziele]

Als einzige Option sei hier `-f` *dateiname* genannt, um *dateiname* anstelle von *Makefile* bzw. *makefile* zu benutzen. Was es mit den optionalen Zielen auf sich hat, wird im Abschnitt 3.3.4.1 erläutert.

Make erzeugt nur Ziele, wenn es notwendig ist, d.h. wenn eine Abhängigkeit jünger als das zugehörige Ziel ist, bzw. das Ziel (noch) nicht existiert. Bei der Verwendung von make zur Programmerzeugung in großen Softwareprojekten kann somit viel Zeit gespart werden, da nur die tatsächlich notwendigen Aktionen (meist Kompilationen) ausgeführt werden.

Sollte eine Abhängigkeit selbst wieder ein Ziel darstellen, so arbeitet make rekursiv.

3.3.4.1 Makefile

Beim Aufruf von make ohne Angabe der Option `-f` wird im aktuellen Verzeichnis nach einer Datei mit Namen `Makefile` oder `makefile` gesucht. Diese Datei beschreibt alle Ziele, Abhängigkeiten und Aktionen.

Ein Makefile kann folgende Einträge besitzen:

- Kommentare
- Makrodefinitionen
- Explizite Regeln
- Implizite Regeln

Kommentare werden durch `#` eingeleitet und gelten bis zum Ende der Zeile.

Makrodefinitionen sind mit Umgebungsvariablen bzw. Variablendefinitionen in Programmiersprachen vergleichbar. Einem beliebig gewählten Makronamen wird ein Wert zugewiesen. Auf den Wert oder Inhalt eines Makros kann dann mittels $(Makroname) (lesend) zugegriffen werden.

Explizite Regeln geben immer eine Zieldatei und, getrennt durch einen Doppelpunkt, deren Abhängigkeiten an, wobei die Abhängigkeiten auch leer sein dürfen. Die Syntax hierfür lautet:

```
Ziel:Abh"angigkeiten
[TAB] Aktion
```

wobei `[TAB]` einen Tabulatorschritt bezeichnet. Ob dieser notwendig ist, oder aber einfache Leerzeichen genügen, hängt von der verwendeten Version von make ab.

Implizite Regeln sollen hier nicht weiter behandelt werden.

Wir stellen nun ein kurzes Beispiel–Makefile vor:

```
# Das ist ein Kommentar
#
# (1)
COMPILER = f95
FLAGS    = -c

# (2)
poly.out: main.o up_1.o up_2.o up_3.o
[TAB] $(COMPILER) main.o up_1.o up_2.o up_3.o -o prog.out

# (3)
```

```
main.o: main.f95
[TAB] $(COMPILER) $(FLAGS) main.f95

# (4)
up_1.o: up_1.f95
[TAB] $(COMPILER) $(FLAGS) up_1.f95

# (5)
up_2.o: up_2.f95
[TAB] $(COMPILER) $(FLAGS) up_2.f95

# (6)
up_3.o: up_3.f95
[TAB] $(COMPILER) $(FLAGS) up_3.f95

# (7)
clean:
[TAB] rm *.o
[TAB] rm prog.out
```

Alle Zeilen, die mit einem #–Zeichen geginnen, sind Kommentarzeilen und haben keinen Einfluss auf die Ausführung von make.

Unter Punkt (1) werden zwei Makros definiert. Das Programm make substituiert bei der Ausführung ein gefundenes Makro mit dem Wert des Makros.

Die Punkte (2) bis (7) stellen explizite Regeln dar.

(2) beschreibt das Ziel *poly.out*. Diese Datei hat eine Reihe von Abhängigkeiten, nämlich *main.o*, *up_1.o*, *up_2.o* und *up_3.o*. Sind alle diese Abhängigkeiten erfüllt, d.h. liegen alle diese Dateien in einer aktuellen Version vor, so wird die Aktion *$(COMPILER) main.o up_1.o up_2.o up_3.o -o prog.out* ausgeführt, wobei hier *$(COMPILER)* durch *f95* ersetzt wird.

Die Punkte (4) bis (6) enthalten als Ziele Objektdateien und als Abhängigkeiten jeweils die Quelldatei. Die Aktion ist jeweils eine Übersetzung der Quelldatei in eine Objektdatei.

Unter (7) sieht man das häufig eingesetztes Ziel `clean`. Durch den Aufruf von `make clean` werden alle Dateien mit der Endung `.o` sowie die Datei `prog.out` im aktuellen Verzeichnis gelöscht.

Beim Aufruf von make kann ein Ziel als Argument angegeben werden, ansonsten wird das erste Ziel im Makefile generiert. Für das obige Beispiel würde also ein Aufruf von make ohne Argumente das Ziel *prog.out* erzeugen. Da dieses Ziel jedoch eine Reihe von Abhängigkeiten besitzt, die selbst wieder Ziele sind, prüft make alle diese Ziele rekursiv. D.h. es werden unter Umständen alle Ziele unter (3) bis (6) erzeugt.

Ein Aufruf von `make clean` für obiges Beispiel würde hingegen nur alle **.o*–Dateien und *prog.out* löschen.

3.4 Zusammenfassung

Unter einem Betriebssystem versteht man die Programme eines Rechners, die zusammen mit den Eigenschaften der Rechenanlage (Hardware) die Grundlage der möglichen Betriebsarten des Rechnersystems bilden. Sie steuern und überwachen die Abwicklung von Programmen.

Betriebsarten sind unter anderem Dialog-, Batch- und Echtzeitbetrieb. Die Hauptaufgaben des Betriebssystems sind Ablaufsteuerung, Arbeitsspeicherverwaltung, Verwaltung der Ein-/Ausgabe und Verwaltung

der Massenspeicher (Platten) und der Dateien.

Zur Unterstützung der Arbeit mit dem Rechner stehen Dienstprogramme zur Verfügung. Typische Komponenten sind Editoren (Programmerstellung, -veränderung: Quellcode), Compiler (übersetzen von Hochsprache in Maschinensprache: Objektcode), Interpreter (befehlsweises Ausführen von Programmen) Linker und Lader (Verbindung von Objektmodulen und Adressberechnung im Arbeitsspeicher), Debugger (Austesten von Programmen, Fehlersuche), Programme zur Dateipflege (Kopieren, etc.) und andere.

Stellvertretend für Betriebssysteme haben wir UNIX näher kennengelernt.

3.5 Fragen und Übungen zu Kapitel 3

F 3.1 Wozu dient das Betriebssystem?

F 3.2 Ist das Betriebssystem eines Computers Hardware oder Software?

F 3.3 Haben wir die PCs im allgemeinen als Batch- oder Dialogrechner benutzt?

F 3.4 Welche Dateiarten gibt es in Unix?

F 3.5 Sei das Dateisystem Abb. 3.2 von S. 35 gegeben. Man ist als Nutzer `hansen` eingeloggt und der Prompt sei `svr50:/home/svr50/hansen>`

(a) Wie wechselt man ins Verzeichnis `/etc`?

(b) Wie kann man sich den Inhalt des Verzeichnisses `texte` anschauen?

(c) Wie kann man sich den Inhalt der Datei `.login` auschauen?

(d) Wie kann man die Datei `brief` ins Verzeichnis `hansen` kopieren?

(e) Wie kann man die Datei `prosa` auf einem Drucker ausgeben?

F 3.6 Gegeben:

```
svr50> ls -alg
drwx--x---   4 hansen edv1     1024 Jun 26 12:48 .
drwx--x--x  13 hansen edv1     1024 Jun 27 14:12 ..
-rw-r-----   1 hansen edv1  8144417 May 31 17:11 aufgaben.zip
-rw-rw----   1 hansen edv1   902640 Jun 26 12:46 aufgabe1.f
drwx-----x   2 hansen edv1     1024 Jun 23 10:13 text
drwx--x--x   2 hansen edv1     1024 Jun 23 10:13 www
drwx------   3 hansen edv1     2048 Jun 24 10:21 TeX
lrwxrwxrwx   1 hansen edv1     1024 May 12 11:38 edv -> /home/stokes/EDV1/
```

Fragen:

(a) Nenne die Dateiarten in dem angegebenen Verzeichnis!

(b) Wer darf das Verzeichnis `www` „betreten“?

(c) Dürfen Mitglieder der Gruppe `edv1` von hier aus `cd edv` aufrufen?

(d) Wer darf die Datei `aufgabe1.f` editieren?

F 3.7 Wie ist der Speicher eines PCs typischerweise organisiert?

F 3.8 Wie ist die allgemeine Kommandosyntax der Shell?

F 3.9 Wie kann man in einem Verzeichnis durch alle Dateien nach dem String ‘text’ suchen und das Ergebnis in der Datei ‘textsuche’ ablegen?

F 3.10 Ist ein Editor Hardware oder Software?

F 3.11 Geben Sie 5 Dienste des Internet an!

F 3.12 Welche Teile gehören zu X–Window?

F 3.13 Beschreiben Sie die Vorgehensweise, wenn man sich auf einem entfernten Unix–Rechner einloggt, dort einen 'Ghostview' startet und die Anzeige auf dem lokalen Rechner geschehen soll!

F 3.14 Wozu dient ein Editor?

F 3.15 Wozu dient ein Compiler?

F 3.16 Wozu dient ein Linker?

F 3.17 Erläutere kurz den Werdegang eines Programmes vom Programmentwurf bis zum ausführbaren Programm. Beschreibe dabei kurz die Funktion und Arbeitsweise der einzelnen beteiligten Dienstprogramme!

3.6 Antworten zu den Fragen und Übungen zu Kapitel 3

A 3.1 s. Skript

A 3.2 Ein Betriebssystem ist Software.

A 3.3 Die PCs werden als Dialogrechner benutzt.

A 3.4 normale Datei (file), Verzeichnis (directory), Link, Gerätedatei (special files), Pipe

A 3.5
(a) cd /etc *oder* cd ../../../etc
(b) ls texte *oder* cd texte [RETURN] ls
(c) more .login *oder* less .login
(d) cp texte/brief ./brief *oder* cp texte/brief . *oder* cp texte/brief ~
(e) lpr texte/prosa

A 3.6 (a)

Dateiname	Dateiart
.	Verzeichnis (current working directory)
..	Verzeichnis (parent directory)
aufgaben.zip	normale Datei (plain file)
aufgabe1.f	normale Datei (plain file)
text	Verzeichnis
www	Verzeichnis
TeX	Verzeichnis
edv	Link

(b) Der Besitzer des Verzeichnisses `hansen` und Mitglieder der Gruppe `edv1`. Andere dürfen nicht, da für `.` anderen (other) keine Rechte gewährt wurden. (Was auch bedeutet, dass das x–Bit für other bei `text` unnötig ist.)

(c) Sie dürfen `cd edv` aufrufen. Ob der Befehl erfolgreich ist, hängt aber von den Rechten der Datei `/home/stokes/EDV1/` ab.

(d) Der Besitzer der Datei `hansen` und Mitglieder der Gruppe `edv1`.

A 3.7 Das Speichersystem eines PCs ist typischerweise hierarchisch organisiert: L1–Cache, L2–Cache, RAM, Swap–Bereich auf der Festplatte.

A 3.8 *kommando–name [-optionen] [parameter]*

A 3.9 more * | grep text > textsuche

A 3.10 Ein Editor ist ein Dienstprogramm des Betriebssystems und somit Software.

A 3.11 z.B. telnet, WWW, ftp, e–mail, news

A 3.12 Server, Client, Kommunikationskanal

A 3.13 Befehlsfolge:
local: xhost +, telnet *host*,
host: Eingabe von Zugangskennung/Passwort, setenv DISPLAY *localhost*:0.0, ghostview &

A 3.14 Ein Editor dient zum Erstellen, Kontrollieren und Verändern von ASCII–Dateien.

A 3.15 Ein Compiler dient zum Übersetzen eines Programms von Hochsprache in Maschinensprache (Objektcode).

A 3.16 Ein Linker dient zum Zusammenfügen von Objektmodulen zu einem lauffähigen Programm unter Berechnung der absoluten Adressen aus den relativen Adressen.

A 3.17 Die nach dem Programmentwurf vorliegenden verfeinerten Fließbilder werden in einer höheren, problemorientierten Programmiersprache formuliert und mit Hilfe des Editors in den Arbeitsspeicher des Rechners eingegeben. Von dort wird es in der Regel über ein Kommando als Datei auf einer Platte abgelegt. Diese Quelldatei wird mit dem Compiler übersetzt, der auch eine Syntaxüberprüfung durchführt und Fehlermeldungen ausgibt. Der Linker verbindet alle Objektmodule zu einem ausfürbaren Programm und stellt dabei die Adressbezüge zwischen ihnen her. Der Lader befördert das Programm in den Arbeitsspeicher des Rechners und ordnet dabei den Symbolen (Namen) absolute Adressen im Arbeitsspeicher zu.

Kapitel 5

Darstellung von Zahlen und Zeichen im Rechner

Bevor wir uns der Darstellung von Zeichen und Zahlen im Detail widmen, müssen wir zuerst verstehen, wie Daten im Rechner allgemein abgelegt werden. Bei Digitalrechnern erfolgt die Informationsdarstellung in binärer Form. Die kleinste Informationseinheit ist das **Bit** (engl. *binary digit*). Es kann zwei Zustände annehmen, die mit 0 und 1 bezeichnet werden. Technisch realisiert werden diese Zustände z.B. durch zwei unterschiedliche Spannungspegel auf einer Signalleitung oder zwei verschiedene Magnetisierungsrichtungen auf einem Informationsträger (Speichermedium). Mehrere Bits werden zu größeren Informationseinheiten zusammengefaßt. So besteht ein Halbbyte (nibble) aus 4 Bits und ein **Byte** aus 8 Bit. Das Byte ist heute in der Regel auch die kleinste adressierbare Speichereinheit (Speicherzelle).

Bei der Darstellung von Informationseinheiten werden die einzelnen Bits von rechts nach links, beginnend mit Null, durchnumeriert. Bit 0 wird als niedrigstwertiges Bit (LSB — engl. **l***east* **s***ignificant* **b***it*), das Bit mit dem höchsten Index als höchstwertiges Bit (MSB — engl. **m***ost* **s***ignificant* **b***it*) bezeichnet:

Byte								1	1	0	0	1	1	0	1	
8 Bit								Bit7=MSB							Bit0=LSB	
Doppelbyte	0	1	1	1	0	0	0	1	0	1	0	0	0	1	0	1
16 Bit	Bit15=MSB															Bit0=LSB

Tabelle 5.1: Darstellung von Informationseinheiten

Da ein Bit genau zwei verschiedene Zustände annehmen kann, gibt es für ein Byte genau $2^8 = 256$ verschiedene Zustände (Bitmuster). Mit Hilfe eines Bytes lassen sich also 256 verschiedene Gegenstände, wie Zeichen, Kommandos oder Zahlen bezeichnen.

5.1 Darstellung von Zeichen

5.1.1 ASCII–Kodierung

Zur Darstellung von Zeichen, wie Buchstaben, Ziffern und Sonderzeichen eines Alphabets, wird in Mikroprozessorsystemen vorwiegend der ASCII (*American Standard Code of Information Interchange*) verwendet. Er ist international genormt (ISO 7–Bit–Code; DIN 66003), in ihm ist jedes Zeichen durch einen 7–Bit–Code dargestellt.

BIT 5–7	0	1	2	3	4	5	6	7
1–4	000	001	010	011	100	101	110	111
0/ 0000	NUL	DLE	SP	0	@	P	‘	p
1/ 0001	SOH	DC1	!	1	A	Q	a	q
2/ 0010	STX	DC2	“	2	B	R	b	r
3/ 0011	ETX	DC3	#	3	C	S	c	s
4/ 0100	EOT	DC4	$	4	D	T	d	t
5/ 0101	ENQ	NAK	%	5	E	U	e	u
6/ 0110	ACK	SYN	&	6	F	V	f	v
7/ 0111	BEL	ETB	’	7	G	W	g	w
8/ 1000	BS	CAN	(	8	H	X	h	x
9/ 1001	HT	EM	)	9	I	Y	i	y
A/ 1010	LF	SUB	∗	:	J	Z	j	z
B/ 1011	VT	ESC	+	;	K	[	k	{
C/ 1100	FF	FS	,	<	L	\	l	|
D/ 1101	CR	GS	−	=	M	]	m	}
E/ 1110	SO	RS	.	>	N	ˆ	n	˜
F/ 1111	SI	US	/	?	O	_	o	DEL

Tabelle 5.2: Zeichentabelle des ASCII mit Hexadezimal und Binärcode

Das höchstwertige achte Bit kann als Paritätsbit genutzt werden. Es ist damit eine Möglichkeit gegeben, Übertragungsfehler zu erkennen. Hierbei wird das MSB jedes Bytes gesetzt oder gelöscht, damit die Anzahl der gesetzten Bits gerade (engl. *even*) oder ungerade (engl. *odd*) ist. Erwartet der Empfänger laut Vereinbarung nur Bytes mit gerader Anzahl Einsen (gesetztes Bit), empfängt aber ein Byte mit ungerader Anzahl, so wird dies als Übertragungsfehler interpretiert.

Verzichtet man auf eine Paritätsprüfung kann das achte Bit zur Erweiterung des Zeichensatzes herangezogen werden. Damit können z.B. länderspezifische Buchstaben wie deutsche Umlaute kodiert werden.

5.2 Zahlensysteme

Bevor wir zur internen Repräsentation von Zahlen im Rechner kommen, wollen wir hier kurz auf unterschiedliche Zahlensysteme eingehen. In der Regel beschränken wir uns im täglichen Leben auf das dezimale Zahlensystem (5.2.1). Alle Zahlensysteme beruhen auf folgendem Bildungsgesetz:

$$\boxed{Z = \sum_i X_i Y^i \; i \in Z \;\; 0 \leq X < Y} \; [6]$$

Dabei ist Y die Basis (dezimal: 10) des Zahlensystems und X der Ziffernvorrat (dezimal: 0...9), dessen größter Wert stets um 1 kleiner als die Basis ist. Der Wert einer Zahl ergibt sich durch Auflösung der Summenformel:

$$\boxed{Z = \ldots X_2 Y^2 + X_1 Y^1 + X_0 Y^0 + X_{-1} Y^{-1} + X_{-2} Y^{-2} \ldots} \; [6]$$

Die negativen Exponenten ergeben die Nachkommastellen.

Wir werden zur Unterscheidung der Zahlen die Basis jetzt immer als Index mit angeben. In der Literatur wird auch B (binär), O (oktal), D (dezimal) oder H (hexadezimal) als Index zur eindeutigen Darstellung benutzt.

5.2.1 Dezimalzahlen

In unserem täglichen Leben sind wir mit dem dezimalen Zahlensystem vertraut. Dabei werden zur Darstellung von Zahlen die zehn Ziffern 0 bis 9 verwendet, die entsprechend ihrer Position mit Zehnerpotenzen gewichtet sind. Der Wert einer Zahl ergibt sich somit entsprechend obiger Formel, z.B.:

$$106_{10} = 1 \cdot 10^2 + 0 \cdot 10^1 + 6 \cdot 10^0 \ .$$

5.2.2 Dualzahlen / Binärzahlen

Im dualen Zahlensystem stehen zur Darstellung von Zahlen lediglich die beiden Dualziffern 0 und 1 zur Verfügung, die entsprechend ihren Positionen mit Potenzen von 2 gewichtet sind.

$$0110\ 1010_2 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 106_{10}$$

Die erste 0 ($0 \cdot 2^7$) trägt nichts zum Ergebnis bei, wird aber zur Anpassung an das 8–Bit–Format (1 Byte) mitgeschrieben. Dualzahlen werden auch als Binärzahlen bezeichnet

5.2.3 Hexadezimalzahlen

Von großer praktischer Bedeutung ist die Stellenschreibweise zur Basis 16, das *Hexadezimalsystem*. Im Hexadezimalsystem gibt es sechzehn Werte. Wir nennen diese 0 bis 9 und A bis F.

$$6A_{16} = 6 \cdot 16^1 + 10 \cdot 16^0 = 106_{10}$$

Um eine Dualdarstellung in eine Hexadezimaldarstellung umzurechnen, betrachten wir unteres und oberes Halbbyte (auch unteres und oberes Nibble genannt) getrennt voneinander und benutzen die Tabelle 5.3

$$106_{10} = \underbrace{0110}_{6}\ \underbrace{1010}_{A} = 6A_{16}$$

Wir können also den gleichen Wert in verschiedenen Zahlensystemen unterscheidlich darstellen. Tabelle 5.3 zeigt dies am Beispiel der Dezimalzahlen von 0 bis 15.

dezimal	binär	hex	dezimal	binär	hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Tabelle 5.3: Dezimale, duale und hexadezimale Zahlen

5.3 Darstellung von Zahlen im Rechner

5.3.1 Natürliche Zahlen

Die natürlichen Zahlen haben immer ein positives Vorzeichen. Aus diesem Grund brauchen wir uns um eine Darstellung nicht zu kümmern. Mit einem Byte können wir also alle Zahlen von $0000\ 0000_2$ bis $1111\ 1111_2$ (0_{10} – 255_{10}) darstellen. Sind zur Speicherung mehrere Bytes vorhanden vergrößert sich der Darstellungsbereich entsprechend.

Anzahl von Bytes	1	2	3	4
größte darstellbare Zahl	255_{10}	$65\ 535_{10}$	$16\ 777\ 215_{10}$	$\approx 4.3 \cdot 10^9$

Tabelle 5.4: Zahlbereiche

In FORTRAN gibt es keinen entsprechenden Datentyp. Wir wollten hier aber der Vollständigkeit halber nicht darauf verzichten, da es in anderen Programmiersprachen Datentypen gibt, die genau diesem Zahlbereich entsprechen.

5.3.2 Ganze Zahlen — INTEGER

Der FORTRAN–Datentyp INTEGER umfaßt die ganzen Zahlen. Damit ergibt sich die Notwendigkeit, bei der internen Repräsentation dieser Zahlen das Vorzeichen zu beachten.

5.3.2.1 Darstellung mit Vorzeichenbit

Eine Möglichkeit der Kodierung von Zahlen mit Vorzeichen ist die Reservierung eines Bits für das Vorzeichen. Das Bit 8 eines Bytes wird mit 1 belegt, wenn die Zahl negativ ist und mit 0 wenn sie positiv ist. Damit stehen für den Betrag der Zahl nur noch 7 Bits zur Verfügung. Damit ergibt sich ein Darstellungsbereich von -128_{10} bis $+127_{10}$. Problematisch ist die Darstellung der 0, da es nun zwei Möglichkeiten gibt $+0$ und -0.

5.3.2.2 Darstellung im Zweierkomplement

Bei der Darstellung im Zweierkomplement erkennen wir negative Zahlen auch an einer 1 im höchstwertigen Bit (MSB), aber es gibt nur eine Darstellung für die Zahl 0. Die Bildung einer negativen Zahl in Zweierkomplementdarstellung erfolgt durch Negation aller Bits der positiven Zahl und anschließender Addition von 1. Ein Beispiel soll das erläutern:

$$+106_{10} = 0110\ 1010_2$$

A 5.1 Negieren der Bits: $0110\ 1010 \rightarrow 1001\ 0101$

A 5.2 Addition von 1: $1001\ 0101 + 0000\ 0001 = 1001\ 0110$

$$1001\ 0110_2 = -106_{10}$$

Die Darstellung von negativen Zahlen im Zweierkomplement hat sich als geeigneter erwiesen als die Darstellung mit Vorzeichenbit und hat somit die größere praktische Bedeutung erlangt.

5.3.2.3 127–Exzeßdarstellung

Die Exzeßdarstellung wird zur Darstellung des Exponenten von REAL–Zahlen verwendet (siehe 5.3.3.1).

$$+105_{10} = 1110\ 1000_2 = E8_{16} \qquad -105_{10} = 0001\ 0110_2 = 16_{16}$$

Der Übergang ähnelt der Zweierkomplementbildung: Negieren aller acht Bit und anschließendes Subtrahieren von eins. Wichtig ist jedoch, daß $+105_{10}$ nicht dieselbe Darstellung hat wie 105 in der vorzeichenlosen Darstellung; vielmehr wird für +105 die vorzeichenlose Darstellung von $232_{10} = E8_{16}$ geschrieben, also eine Zahl mit dem Überschuß (Exzeß, engl. *bias*) 127_{10}. Entsprechend steht für -105_{10} die vorzeichenlose Darstellung von $-105_{10} + 127_{10} = 22_{10} = 16_{16}$. Die Exzeßdarstellung wird zur Darstellung des Exponenten von REAL–Zahlen verwendet (siehe weiter unten bei der Mikroprozessornorm).

5.3.3 Reelle Zahlen — REAL

Wir haben bisher den Zahlentypen INTEGER kennengelernt. Nachkommastellen lassen sich nur in der Menge der reellen Zahlen ausdrücken. Für diese Menge stehen in FORTRAN die Zahlentypen REAL und DOUBLE PRECISION zur Verfügung. Sie unterscheiden sich in der Genauigkeit der Darstellung der Zahlen. Wir können in einem Rechnersystem nur eine endliche Zahl von Nachkommastellen abspeichern und müssen uns somit auf eine sinnvolle Rechengenauigkeit beschränken. Die Wahl des Datentyps hängt von verschiedenen Faktoren der Problemstellung ab: Welche Genauigkeit haben die Eingangsdaten? Benutzen wir eine iterative Rechenvorschrift, die eine höhere Genauigkeit erfordert?

5.3.3.1 REAL–Zahlen nach Mikroprozessornorm

Für die Darstellung von REAL–Zahlen im Rechner gibt es sehr verschiedene Konventionen. Die hier vorgestellte Mikroprozessornorm (*IEEE Standard P754 - 1985*) arbeitet mit Darstellungen zur Basis 2.

- **Vorzeichen**: Ein von der Mantisse getrenntes Bit dient zur Darstellung des Vorzeichens der Mantisse. Es ist gesetzt, wenn das Vorzeichen der Mantisse minus ist.
- **Mantisse**: Zur Darstellung der Mantisse dienen 24 Bit, normalisiert auf die Werte 1.0_2 bis $1.11...1_2$. Es handelt sich um Dualbrüche, die durch den Dualpunkt markiert sind: es ist $1.1_2 = 1\frac{1}{2} = 1.5_{10}$ und $1.01_2 = 1\frac{1}{4} = 1.25_{10}$. Die auf 1.0_2 folgende REAL–Zahl ist also $1.00..01_2$ mit der letzten 1 in der Position für 2^{-23}. Somit ist $\varepsilon = 2^{-23} \approx 1.2 \cdot 10^{-7}$.

  Da die Mantisse stets normiert ist, ist das höchstwertige Bit stets zu eins gesetzt. Es ist daher nicht notwendig, den Wert dieses Bits mit abzuspeichern. Von den 24 Bits der Mantisse werden nur die 23 niedrigwertigen Bits abgespeichert, das höchstwertige Bit wird „mitgedacht“, aber nicht abgespeichert, es heißt daher verstecktes Bit (engl. *hidden bit*).
- **Exponent**: Acht Bit dienen zur Darstellung des Exponenten zur Basis 2, und zwar in der 127–Exzeß–Darstellung. Den Wert des Exponenten erhält man durch Abziehen von 127 von dem in den 8 Bit dargestellten ganzzahligen Wert ohne Vorzeichen E' (siehe Tabelle 5.5).

  Dabei ist der Wert des Exponenten +128 reserviert für die Darstellung von $\pm\infty$, bzw. ungültiger REAL–Werte (NaN = engl. **n***ot* **a n***umber*). Der größtmögliche Exponent ist also +127, und 2^{127} ist etwas größer als 10^{38}. Ein abgetrenntes Vorzeichenbit für den Exponenten gibt es nicht; allenfalls läßt sich das höchstwertige Bit von E' als invertiertes Vorzeichenbit auffassen.

Die Abspeicherung erfolgt nach dem Schema in Abbildung 5.1: Das höchstwertige Bit 31 nimmt das Vorzeichen für die Mantisse auf, in Bit 30 bis Bit 23 steht der Wert E', der Exponent in 127–Exzeßdarstellung, in Bit 22 bis Bit 0 die Werte $b_1 b_2 \cdots b_{23}$ der Mantisse. Links von b_1 ist der Dualpunkt, davor eine 1 zu denken.

E' binär	E' dezimal	E dezimal
$0000\ 0000_2$	0	-127
$0000\ 0001_2$	1	-126
$\cdots$	$\vdots$	$\vdots$
$0111\ 1111_2$	127	0
$\cdots$	$\vdots$	$\vdots$
$1111\ 1110_2$	254	$+127$
$1111\ 1111_2$	255	∞ oder NaN

Tabelle 5.5: Darstellung des Exponenten bei REAL–Zahlen

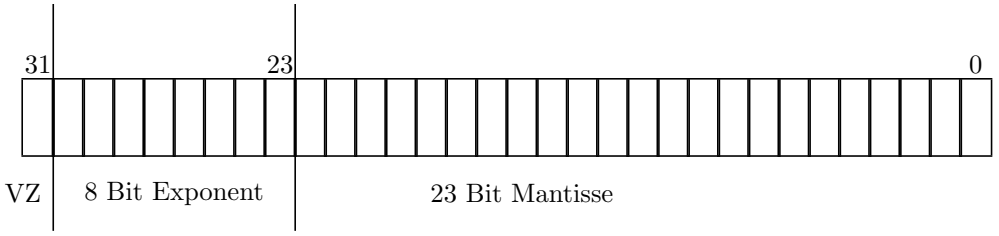


Abbildung 5.1: Abspeicherung einer REAL–Zahl nach Mikroprozessornorm

5.3.3.2 Erhöht genaue Zahlendarstellung

Da die Genauigkeit im Laufe umfangreicher Rechnungen unweigerlich leidet, reicht die oben beschriebene REAL–Darstellung nicht immer aus. Die Mikroprozessornorm kennt daher außer der obigen, als *single precision* bezeichneten Darstellung eine weitere, die als *long* oder **DOUBLE PRECISION** bezeichnet wird; der deutsche Ausdruck ist „doppelt genau“. Diese Darstellung verwendet acht Byte (64 Bit), die nach einem Schema aufgeteilt werden, das in Abbildung 5.2 dargestellt ist[1].

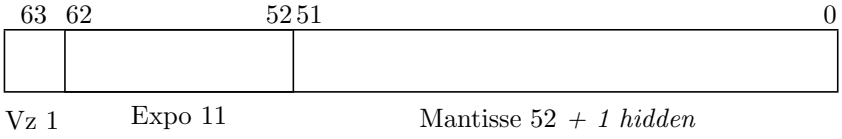


Abbildung 5.2: Abspeicherung einer doppelt genauen Zahl

Wir erläutern lediglich die Unterschiede zur vorher besprochenen REAL–Darstellung:

- **Vorzeichen**: Bit 63
- **Exponent**: 11 Bit dienen zur Darstellung des Exponenten zur Basis 2, und zwar in der 1023–Exzeßdarstellung. Die gültigen Exponenten reichen von -1023 bis $+1023$, und 2^{1023} ist etwa 10^{308}.
- **Mantisse**: Zur Darstellung der Mantisse dienen 53 Bit, normalisiert auf die Werte 1.0 bis $1.11\cdots1_2$. Es werden nur die 52 niedrigwertigen Bits abgespeichert. 1.0 unterscheidet sich von der nächstfolgenden Zahl $1.0\cdots01_2$ um den Wert $\varepsilon = 2^{-52} \approx 2.2 \cdot 10^{-16}$.

[1] Es gibt noch weitere Möglichkeiten der Abspeicherung

In den neueren, modernen Programmiersprachen (z.B. C++, F95) ist es außerdem möglich, die Anzahl der Bytes und somit die Genauigkeit frei zu wählen. (IEEE Standard P854)

5.3.3.3 Genauigkeit der internen Zahlenrepräsentation

Reelle Zahlen können auf verschiedene Weise geschrieben werden. Ein Beispiel:

$$-0.0031415^{+03} \qquad -0.0314159^{+02} \qquad -0.3141592^{+01}$$

Dabei ist erkennbar, daß wir mit der letzten Darstellung das höchste Maß an Genauigkeit erzielen können. Deshalb wird auch bei der internen Zahlendarstellung im Rechner die Mantisse stets auf einen Wert zwischen 1.00... und 9.99... normalisiert. Die Zahl 0.0 läßt sich nicht auf diese Weise normalisieren. Es gibt dafür eine Ausnahme.

Mit der begrenzten Menge an Nachkommastellen für die interne Darstellung von reellen Zahlen haben wir auch nur eine endliche Anzahl möglicher Werte. Da Rechnungen typischerweise ein Ergebnis liefern, das nicht genau darstellbar ist, müssen die Ergebnisse zum Weiterverarbeiten durch eine darstellbare REAL–Zahl ersetzt werden. Dafür gibt es zwei Möglichkeiten — Rundung oder Abschneiden.

5.3.4 BCD–Kodierung dezimaler Zahlen

Der BCD–Code (Binär–Dezimal–Code, engl. **b***inary* **c***oded* **d***ecimal*) stellt die Ziffern von 0 bis 9 durch die ihnen entsprechenden Dualzahlen in jeweils 4 Bit dar (Tabelle 5.6). Diese Kodierung großer Dezimalzahlen ist von Bedeutung, da viele Prozessoren diesen Code in ihrem Befehlssatz direkt unterstützen.

Dezimal-ziffer	BCD–Code
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
Stellen-wert	8 4 2 1

Tabelle 5.6: Der BCD–Code

Ein Beispiel soll das Verfahren verdeutlichen: Die Zahl 145 933 911_{10} soll kodiert werden. Es werden genau 5 Byte = 40 Bit dafür benötigt.

$$\underbrace{01}_{0000\ 0001}\ \underbrace{45}_{0100\ 0101}\ \underbrace{93}_{1001\ 0011}\ \underbrace{39}_{0011\ 1001}\ \underbrace{11}_{0001\ 0001}$$

Der Vorteil gegenüber der Darstellung als Zeichenkette liegt in dem benötigtem Speicherplatz, da für jedes Zeichen ein Byte gebraucht wird, müßten für die Zahl mit neun Ziffern 9 Byte bereitgestellt werden.

5.4 Zusammenfassung

Die kleinste Informationseinheit ist das Bit, es kann zwei Zustände annehmen. Größere Einheiten sind das Byte (8 Bit) und das Wort (4 Byte).

Der ASCII ist ein 7–Bit–Code für die Darstellung von alphanumerischen, Sonder- und Steuerzeichen, das achte Bit kann für Erweiterungen oder zur Übertragungssicherung genutzt werden.

In der Informationstechnik sind neben der Dezimaldarstellung von Zahlen auch noch die Zahlensysteme zur Basis 2 (Dual- oder Binärzahlen) sowie zur Basis 16 (Hexadezimalzahlen) von Bedeutung. Ein Wert kann somit in den unterschiedlichen Zahlensystemen auf verschiedene Art und Weise dargestellt werden.

Es gibt verschiedene Methoden, um ganze Zahlen mit Vorzeichen (Typ INTEGER) darzustellen. Konventionen mit zwei verschiedenen Darstellungen für die Null haben sich nicht bewährt. Allgemein benutzt wird die Zweierkomplementdarstellung mit dem unsymmetrischen Zahlenbereich -128_{10} bis $+127_{10}$. Die Exzeßdarstellung findet bei der Exponentendarstellung von REAL–Zahlen Anwendung.

Rechnen im Zweierkomplement wird für vorzeichenlose Zahlen wie für INTEGER in völlig gleicher Weise durchgeführt. Zur Markierung von Grenzwertverletzungen dienen in den Mikroprozessoren zwei Register, das C–Flag (Übertrag, Carry) für natürliche Zahlen, 0_{10} bis 255_{10}, und das V–Flag (Überlauf, oVerflow) für den Typ INTEGER, -128_{10} bis $+127_{10}$.

Eine REAL–Zahl wird über das Vorzeichen, die Mantisse und den Exponenten dargestellt.

Die Mikroprozessornorm stellt REAL–Zahlen mit 32 Bit oder 64 Bit dar: ein Bit für das Vorzeichen der Mantisse, acht bzw. elf Bit für den Exponenten in Exzeß–Darstellung, 24 bzw. 53 Bit für die Mantisse. Da die Mantisse stets normalisiert wird, ist es nicht notwendig, ihr höchstwertiges Bit mit abzuspeichern (engl. *hidden bit*).

Zur komprimierten Darstellung von Dezimalzahlen wird meistens der BCD–Code verwendet, bei dem jede Dezimalziffer in vier Bit gespeichert sind.

Der Rechner muß also aufgrund der unterschiedlichen internen Darstellungen wissen, wie er eine im Speicher stehende Bitfolge interpretieren muß: ob als Zeichenfolge, als ganze Zahl oder als reelle Zahl. Führen wir uns die unterschiedlichen Interpretationen an einem Beispiel vor Auge.

Gegeben sei ein Wort (32–Bit) mit dem Inhalt

$$0010\ 0001\ 0100\ 1000\ 0011\ 1100\ 0111\ 1000_2$$

In Hexadezimalschreibweise sind das $21483C78_{16}$ und als Realzahl interpretiert bedeutet der Inhalt $+6.78... \cdot 10^{-19}$. Als Integer*4 ergibt sich $558\ 382\ 200_{10}$ und als CHARACTER*4 erhalten wir die Zeichenfolge `'!H<x'`.

5.5 Fragen und Übungen zu Kapitel 5

F 5.1 Wieviele verschiedene Zeichen kann man mit 16 Bit darstellen?

F 5.2 Wieviel Bit hat ein Byte?

F 5.3 Was ist hexadezimal A4 in Dezimalschreibweise?

F 5.4 Welche Darstellung für Integerzahlen hat sich bei Computern durchgesetzt?

F 5.5 Aus welchen drei Komponenten besteht eine REAL–Zahl?

F 5.6 Was ist der BCD–Code und wo wird er benutzt?

F 5.7 Wie lautet die Dual-, Oktal- und Hexadezimaldarstellung von 105_{10}?

F 5.8 Was ist ASCII?

F 5.9 Wie groß ist der Unterschied zwischen den FORTRAN–Konstanten 1.6EO und 1.6DO?

5.6 Antworten zu den Fragen und Übungen zu Kapitel 5

A 5.1 Mit 16 Bit kann man $2^{16} = 65536$ verschiedene Zeichen darstellen.

A 5.2 Ein Byte hat 8 Bit.

A 5.3 $A4_{16} = 10 \cdot 16^1 + 4 \cdot 16^0 = 164_{10}$

A 5.4 Für Integerzahlen wird die Zweierkomplementdarstellung benutzt.

A 5.5 Vorzeichen, Mantisse und Exponent

A 5.6 Der BCD–Code ist ein Code zur kompakten Darstellung von Dezimalzahlen. Jede Dezimalziffer wird in einem Halbbyte gespeichert. Der BCD–Code wird hauptsächlich im Bereich der kommerziellen Datenverarbeitung (Geschäftsbereich) verwendet.

A 5.7 $01101001_2 = 151_8 = 69_{16} = 105_{10}$

A 5.8 American Standard Code for Information Interchange, ein international genormter 7–Bit Code für alphanumerische, Sonder- und Steuerzeichen.

A 5.9 Der Unterschied zwischen den FORTRAN–Konstanten 1.6E0 und 1.6D0 beträgt etwa 10^{-7}, die Differenz der beiden Maschinenepsilon für REAL ($\varepsilon \approx 1.2 \cdot 10^{-7}$) und DOUBLE PRECISION ($\varepsilon \approx 2.2 \cdot 10^{-16}$).

Kapitel 6

Rechner – Aufbau

6.1 Aufgaben eines Rechners

Allgemein ist ein Rechner (engl. *Computer*) ein Gerät zum Rechnen. Im Laufe der Zeit hat sich dieser Begriff dahingehend verändert, daß man unter „Rechner“ ein komplexes Gerät versteht, das alle möglichen Anwendungen neben dem Rechnen an sich durchführt:

- Lösung von numerisch–mathematischen Problemen, besonders im Ingenieurbereich: Prozeß- und System–Simulation, computerunterstützter Entwurf von Fahrzeugen, Bauteilen und elektronischen Schaltkreisen (CAD/CAM), meist grafikgestützt.
- Grafische Anwendungen wie Darstellung des Aufbaus von Atomen und Molekülen, Visualisierung allgemeiner Meß- und Simulationsergebnisse.
- Steuerung und Regelung industrieller Prozesse, wie Fertigungsstraßen für Automobile oder chemische Produktionsverfahren.
- Verwaltung und Auswertung großer Informationsmengen (Datenbanken, Statistik).
- Be- und Verarbeitung von Daten (Tabellenkalkulation), Codierung, Chiffrierung, Übersetzung; Textverarbeitung und Desktop–Publishing.
- Spiele, Musik

Die Liste kann noch um viele weitere wichtige Anwendungen erweitert werden. Zur Bearbeitung solch unterschiedlicher spezieller Aufgaben benötigt der Rechner ein Programm, das ihn zu einer Spezialmaschine macht. Andererseits wird nicht jeder Rechner für alle Aufgaben gleichermaßen gut geeignet sein. Tabelle 6.1 zeigt, welche Kategorien von Rechnern sich für welche Aufgaben herausgebildet haben. Man beachte, daß die Leistungsgrenzen in der Abbildung fließend sind. Dank der Fortschritte in der Mikroelektronik hat sich z.B. der Abstand zwischen Workstation und Supercomputer immer weiter verringert.

In Kapitel 3 haben wir gesehen, daß ein Programm — das Betriebssystem — auf dem Rechner abläuft, um die stark rechnernahen Operationen vor dem Benutzer zu verbergen und ihm so den Umgang mit dem Rechner zu erleichtern.

Ein Verständnis für diese rechnernahen Funktionen entwickelt man erst, wenn man den internen Aufbau eines Rechners kennengelernt hat. Wir wollen hierfür von einem modernen Rechnerarbeitsplatz der unteren Kategorie ausgehen und uns dann immer tiefer in den Rechner hineinbegeben.

Super- und Großrechner (mainframes)	CRAY, CONVEX CYBER IBM 30XX	Rechenintensive numerische Anwendungen, Rechenzentren
Minirechner (workstations)	IBM Serie 3 VAX, DEC, HP, SUN, SGI	Anwendungen mittlerer bis großer Rechenleistung, Grafik (CAD, CAM, CAE), künstliche Intelligenz, verteilte Systeme
Mikrorechner	PDP-11 VME-Systeme	Prozeßsteuerung, Echtzeitanwendungen
Personal Computer	IBM PCs Apple Mac	Office–Anwendungen (z.B., Grafik- und Textverarbeitung, Desktop Publishing, Kalkulation), Multimedia–Anwendungen, Spiele, usw.

Tabelle 6.1: Kategorien von Rechnern und ihre Anwendungen.

So ein Rechner ist grob gesprochen eine Kiste mit Bildschirm, Tastatur und Maus. Die Tastatur dient zur Eingabe von Text oder Steuerzeichen (z.B. Positionierung des Cursors auf dem Bildschirm). Der Bildschirm dient der Darstellung von Informationen, die der Rechner ausgibt. Der Cursor markiert die Stelle der Texteingabe, ein Pfeil die Position der Maus. Mit der Maus können bei grafischen Benutzeroberflächen Dienste angewählt und gestartet werden (z.B. Anwählen und Bewegen grafischer Objekte; Editor–Menüs etc.).

Abbildung 6.1: PC — Personal Computer

6.2 Rechner–Arbeitsplatz

Als Beispiel für einen solchen Arbeitsplatz betrachten wir einen Personal Computer ähnlich der Art, wie wir ihn in unseren Übungen benutzen.

Sehen wir uns etwas genauer an, was sich in dem Gehäuse eines Computers verbirgt. Die wichtigsten Komponenten eines Rechners sind

- die Grundplatine (engl. *motherboard*) mit
  - Anschluß für Disketten- und Festplattenlaufwerk(e)
  - Tastaturschnittstelle
  - serielle und parallele Schnittstellen, USB (*Universal Serial Bus*)
  - Arbeitsspeicher (Hauptspeicher)
  - Steckplätze für Erweiterungskarten
  - Controler für die Laufwerke
- Erweiterungskarten
  - Grafikkarte
  - Netzwerkkarte
  - Soundkarte
  - weitere “Controler” (SCSI-Controler, Firewire, usw.)

- Diskettenlaufwerk
- Festplatte
- CD–ROM–/DVD–Laufwerk
- CDR(W)–Brenner, DVD–Brenner
- Netzteil und Lüfter

Die Rechner in unserem Micropool haben keine Soundkarte und kein CD–ROM Laufwerk.

Das wichtigste Element ist die Grundplatine. Daran werden alle Komponenten über Steckplätze oder Flachkabelverbindungen angeschlossen. Die Verbindung der einzelnen Teile des Systems wird durch den **Systembus** hergestellt.

Der Prozessor oder **CPU** (engl. **c***entral* **p***rocessing* **u***nit*), die den Kern des Rechners darstellt, befindet sich i.a. auf der Grundplatine.

Karten sind dünne, rechteckige Plastikplatten mit zahlreichen Leiterbahnen aus Kupfer, die die Sockel der Bausteine verbinden. Von diesen, auch **Chips** genannt, sieht man nur die dunklen, rechteckigen Plastikgehäuse mit den zahlreichen silbernen Kontaktfüßchen. Die eigentlichen Chips, Siliziumplättchen von 1–100 mm^2 sind meist nicht sichtbar. Sie enthalten bis zu Millionen (z.B. Megabit–Chips) von Schaltkreisen, die als Rechner oder Speicher dienen.

Das Netzteil dient der Stromversorgung aller Komponenten und der Lüfter sorgt dafür, daß keine Schäden durch Überhitzung entstehen. Mittlerweile haben die CPU und auch moderne Grafikkarten selbst noch einen Kühlkörper und Lüfter zur Wärmeableitung.

In den folgenden Abschnitten werden wir uns die einzelnen Einheiten genauer ansehen.

6.3 Motherboard

Die CPU ist das Kernstück eines Rechners und führt die Hauptaufgaben aus, die an den Rechner gestellt werden. Für die Ausführung von Programmen im Rechner ist das Zusammenspiel folgender Komponenten unbedingt erforderlich (s. Abb. 6.2):

- Der **Prozessor** führt Rechen- und andere Operationen durch. Moderne Prozessoren sind in einem einzigen Chip integriert (Mikroprozessor). Der Prozessor alleine ist nicht arbeitsfähig, er benötigt Arbeitsspeicher für Programm und Daten.
- Der **Arbeitsspeicher** enthält das Programm (Befehle für den Prozessor) und Daten (Zahlenwerte, die als Operanden in die Rechenoperationen des Prozessors eingehen und als Ergebnisse vom Prozessor in den Arbeitsspeicher geschrieben werden). Programmspeicher und Datenspeicher werden in der Regel strikt getrennt verwaltet.
- Der **Prozessorbus** verbindet Prozessor und Hauptspeicher. An ihn sind auch die Tastatur, der Festplatten- und Diskettenanschluß sowie weitere Steckplätze angeschlossen. Bei einfacheren Rechnern ist der Systembus nur die Verlängerung des Prozessorbusses. Anspruchsvollere Systeme trennen die beiden, um eine schnellere Funktion des Prozessorbusses zu ermöglichen.
- Ein externer **Taktgenerator** (engl. *clock*) erzeugt einen Takt mit einer bestimmten Frequenz (Taktrate), die angibt, mit welcher Geschwindigkeit der Prozessor einen Befehl abarbeitet. Das ist notwendig, um den Prozessor in Betrieb zu halten. Dabei erhält er eine Folge von digitalen Impulsen, die er intern an seine Funktionsgruppen weiterleitet. Taktfrequenzen heute gängiger Prozessoren erreichen bereits Bereiche um die 2 GHz, aber wie schon in Kapitel 1 dargestellt, gibt es hier ständige Weiterentwicklungen.

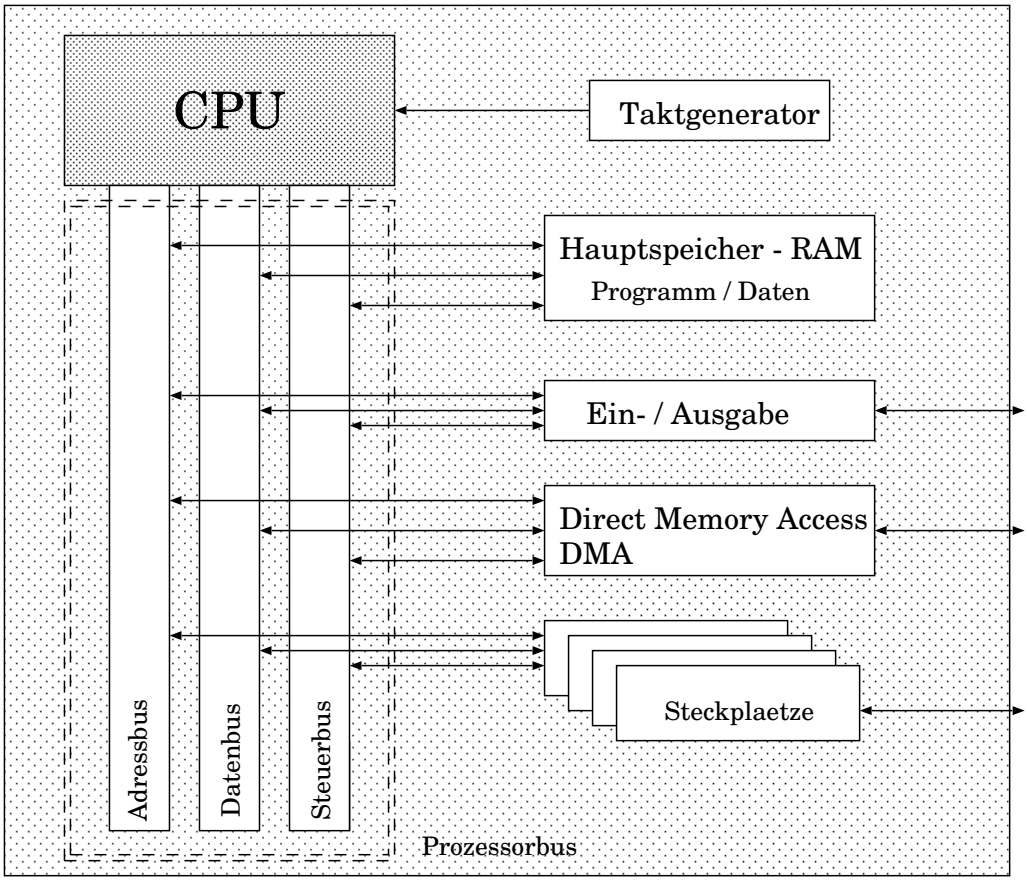


Abbildung 6.2: schematischer Aufbau eines Motherboards

- Die **Ein-/Ausgabe–Einheit** steht in Verbindung mit den Ein-/Ausgabegeräten, wie z.B. Tastatur und Monitor. Sie erledigt die Kommunikation zwischen dem Rechner und seiner Umwelt.
- Die **DMA–Einheit** sorgt für schnellen und effizienten Datentransfer zwischen Hauptspeicher und externen Speichern wie Festplatte oder CD–ROM Laufwerk, ohne den Prozessor dabei mit Arbeit zu belasten. In Abb. 6.2 nicht eingetragen ist eine batteriebetriebene Uhr.

Im folgenden betrachten wir die Funktionseinheiten der Grundplatine genauer.

6.3.1 Der Prozessor

Dieser Abschnitt bringt eine kurze Darstellung des prinzipiellen Aufbaus von Prozessoren. Aufbau und Funktion von Mikroprozessoren werden ausführlich in Kapitel 7 besprochen.

Der Kern der Grundplatine ist der Prozessor. Er arbeitet alle Befehle ab, die ihn über den Datenbus erreichen. Der Prozessor selbst ist in verschiedene Funktionsblöcke unterteilt, die die unterschiedlichen Aufgaben des Prozessors bewältigen, Abb. 6.3.

Ein Prozessor besteht aus einer arithmetisch–logischen Einheit (**ALU**, engl. **a***rithmetic and* **l***ogical* **u***nit*) in Verbindung mit einem Zähler und einem Satz von Registern, die alle Daten des Prozessors aufnehmen:

- Datenregister
- Adreßregister
- Befehlsregister
- Befehlszähler.

Ein Register ist ein schneller Zwischenspeicher, in den Zahlenwerte eingetragen und wieder entnommen werden können. Auf Register kann der Prozessor wesentlich schneller zugreifen als auf Speicherzellen (Faktor > 10). Den Prozessor kann man in vier Bereiche unterteilen:

1. Der erste Bereich dient zur Bearbeitung von Daten und besteht aus der ALU und den Datenregistern. Er wird Datenwerk genannt.
2. Der zweite Bereich dient der Adreßberechnung und besteht aus dem Befehlszähler und den Adreßregistern. Er wird Adreßwerk genannt.
3. Der dritte Bereich dient der Befehlsausführung und besteht aus dem Befehlsregister und dem Befehlsdekodierer, mit dem alle Bausteine über Steuerleitungen angeschlossen sind. Er wird Steuerwerk genannt.
4. Der vierte Bereich dient der Busbehandlung des Prozessors. Hier wird dafür gesorgt, daß ankommende Signale korrekt in den Prozessor übernommen oder nach außen gegeben werden.

Die Gesamtheit der Befehle, die ein Prozessor ausführen kann, nennt man seinen **Befehlssatz** (engl. *instruction set*). Die Befehle lassen sich in folgende Klassen einteilen (siehe auch Kapitel 7):

- Speicher–Operationen: Diese Operationen ermöglichen den Datenaustausch zwischen den verschiedenen Registern und dem Hauptspeicher.
- Arithmetische Operationen: Der Prozessor führt Operationen wie Addition oder Subtraktion aus.

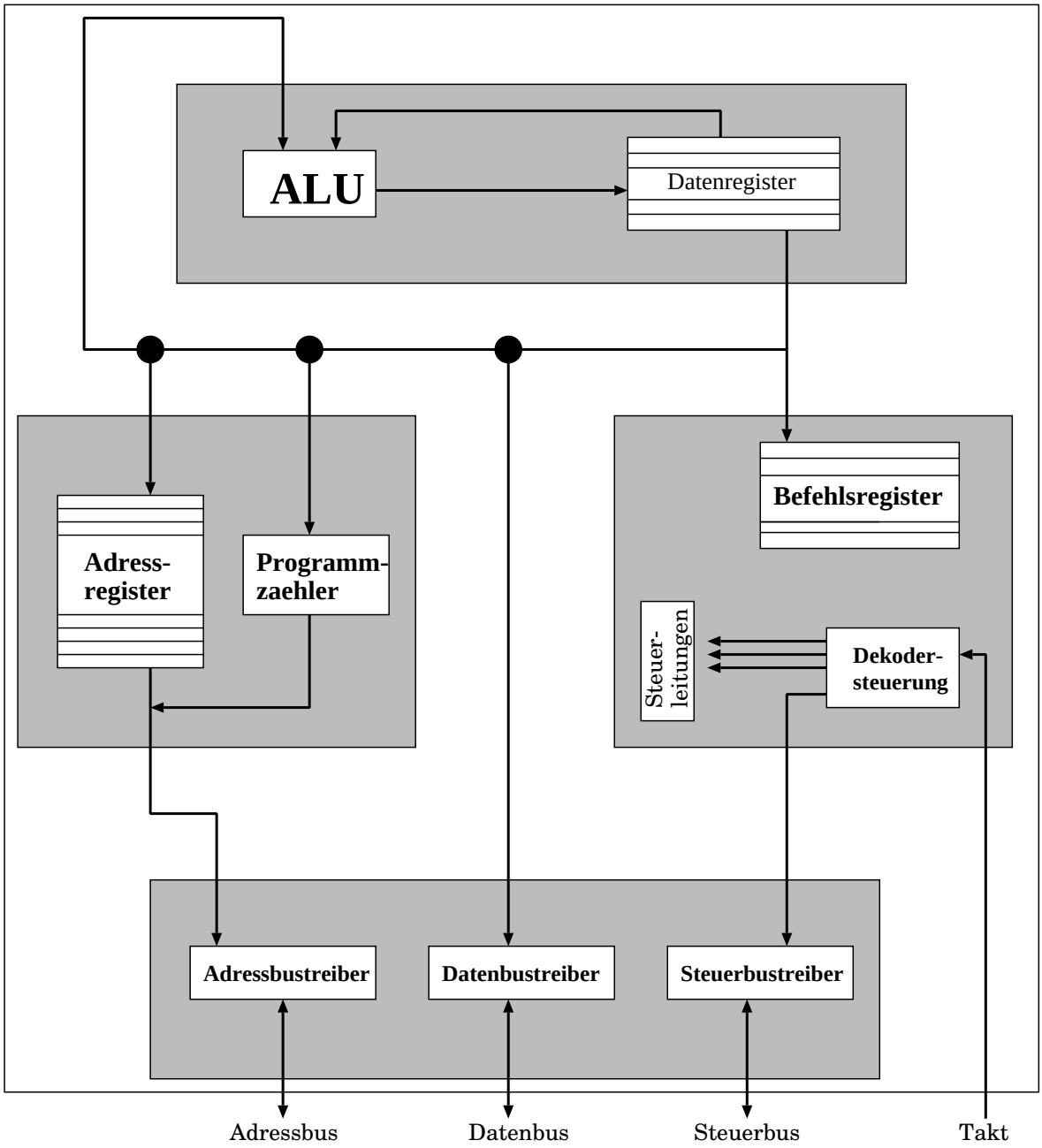


Abbildung 6.3: vereinfachter schematischer Aufbau eines Prozessors

- Logische Operationen: Boolesche Operationen zum logischen Verknüpfen von Daten.
- Sprungoperationen: Ermöglichen Verzweigungen in Programmen. Es gibt zwei Arten von Sprüngen: unbedingte Sprünge und bedingte Sprünge an Adressen.

Aufbau und Funktion eines Mikroprozessors werden im Detail in Kapitel 7 beschrieben.

6.3.2 Der Arbeitspeicher

Der Speicher des Rechners dient dazu, Daten und Befehle des Programmcodes abzulegen, so daß der Prozessor auf diese zugreifen kann. Sie werden im Speicher binär dargestellt, das heißt als Zahlen des Dualsystems. Diese Zahlenwerte befinden sich in Speicherzellen, die jeweils über eine eigene **Adresse** angesprochen werden können.

In der Regel ist beim Speicher die kleinste adressierbare Speicherzelle ein **Byte**. Für große Zahlenwerte reicht ein Byte nicht aus. Man verwendet dann ein Doppelbyte (zwei aufeinanderfolgende Bytes), oft als **Wort** bezeichnet, und ein **Doppelwort** (vier aufeinanderfolgende Bytes), auch als Langwort bezeichnet.

Die Größe des Speichers, den ein Rechner hat, oder bis zu der er maximal ausgebaut werden kann, ist ein wichtiges Leistungsmerkmal. Die **Speichergröße** wird meist in Megabyte (MByte, MB) angegeben.

Die Vorsilbe „Kilo–“ bedeutet im Computerbereich $2^{10} = 1024$, „Mega–“ entsprechend $1024 \cdot 1024 = 1.048.576$, und „Giga–“ entspricht $1024^3 \approx 1,074 \cdot 10^9$.

Die maximale Speichergröße, die der Prozessor verwalten kann, wie auch die Größe des eingebauten Arbeitsspeichers, sind stets Zweierpotenzen. So sind beispielsweise $2^{16} = 65.536 = 64$ kB oder $2^{24} = 16.777.216 = 16$ MB. Die Adressen des Speichers werden von Null bis zum größtmöglichen Wert durchnumeriert (vgl. dazu den Datentyp NATURAL in Kap. 5). Der größtmögliche Adreßwert bestimmt wesentlich die Breite des Adreßbusses, der Adreßregister und des Befehlszählers.

Der Zugriff auf die Befehle und Daten erfolgt über die Adresse, an der der entsprechende Zahlenwert für den Befehl oder für das Datum im Speicher steht. Im Prinzip ist jede Speicherzelle gleichschnell zugreifbar, der Speicher verfügt über **wahlfreien Zugriff** (engl. *RAM* — **r***andom* **a***ccess* **m***emory*).

Auf der Grundplatine befindet sich der Hauptspeicher in Form von Speicherbausteinen (Speichermodule). Es gibt zwei Arten von RAM–Bausteinen, statisches RAM, das die in ihm abgespeicherte Information behält, solange die Versorgungsspannung anliegt, und dynamisches RAM, in dem zusätzlich die abgespeicherte Information immer wieder in kurzen Abständen elektrisch aufgefrischt (engl. *refresh*) werden muß. Nach Ausschalten der Versorgungsspannung geht in beiden RAM–Typen die Information verloren. Wegen des niedrigeren Preises wird dynamisches RAM häufiger eingesetzt als statisches RAM. Die Zugriffszeiten auf den Arbeitsspeicher liegen mittlerweile im Bereich von wenigen Nanosekunden.

Die hohe Integration von Schaltkreisen auf den Speicherchips ermöglicht heute Bausteine mit großen Speichermengen. Der Hauptspeicher auf dem Board läßt sich heutzutage in Gigabytebereiche ausbauen.

Für Spezialzwecke (z.B. beim Starten des Rechners) wird ein Speicher benötigt, der ein Programm und Daten in unveränderbarer Form enthält. Ein solcher Festwertspeicher wird meist als **ROM** (engl. **r***ead–* **o***nly* **m***emory*, Nur–Lese–Speicher) bezeichnet.

6.3.3 Der DMA–Kanal

Die DMA–Schnittstelle (DMA — engl. **d***irect* **m***emory* **a***ccess*) wurde entwickelt, um beim Datentransfer zwischen Arbeitsspeicher und Festplatte/Diskette den Prozessor nicht zu belasten. Um zu vermeiden, daß

der Prozessor auf den Prozessorbus zugreift während man die Daten aus dem Speicher holt, unterbindet man einfach den Takt des Prozessors und stiehlt ihm somit einen Taktzyklus (engl. *cycle stealing*).

Während der Prozessor stillsteht, kann ein Schreib- oder Lesezyklus über den Prozessorbus abgewickelt werden. Dieser Datentransfer kann sehr viel höher sein, als eine vom Prozessor selbst vorgenommene Übertragung. Der intelligente Baustein, der diese Übertragung steuert (**DMA–Controler**) besitzt eigene Register, die zur Vorbereitung des DMA–Transfers vom Prozessor aus geladen werden.

6.3.4 Die Tastaturschnittstelle

Die Eingaben, die durch die Tastatur (engl. *keyboard*) erfolgen, werden meistens von einem integrierten Baustein in der Tastatur selbst bearbeitet. Die Signale werden dann über das Kabel der Tastatur an den Rechner weitergeleitet, wo die Daten von der Keyboardschnittstelle übernommen und dem Keyboardtreiber des Betriebssystems zur Verfügung gestellt werden. Dort wird die Eingabefolge entschlüsselt und in einem Puffer zwischengespeichert.

6.3.5 Der Prozessorbus

Vom Prozessorbus, das sind die parallelen Leitungen auf dem Mainboard zwischen Prozessor und Hauptspeicher, hängt wesentlich die Leistungsfähigkeit des Systems ab, denn es ist der Engpaß, durch den alle Befehle und Daten hindurchlaufen müssen. Wichtig ist einerseits die Breite des Busses, d.h. die Anzahl der Bits, die parallel übertragen werden können, und andererseits die Taktfrequenz. Auf der Grundplatine lassen sich folgende Teil–Busse unterscheiden:

- **Datenbus**: Seine Breite wird durch die maximale Größe der Daten, die der Prozessor mit einem Mal aufnehmen kann, bestimmt. Üblich sind 32 Bit, 64 oder 128 Bit. Über den Datenbus werden Befehle aus dem (Programm-) Speicher zum Prozessor geschickt und Daten (im engeren Sinne) vom (Daten-) Speicher zum Prozessor und zurück zum Speicher. Auch die Daten von Ein-/Ausgabevorgängen gehen über den Datenbus, z.B. von der Festplatte zum Speicher (vgl. Abschnitt 6.3.3, DMA–Kanal). Der Datenbus wird also zum Transport in beiden Richtungen benutzt.
- **Adreßbus**: Seine Breite legt den Bereich fest, den der Prozessor adressieren kann, üblich sind 16, 32, oder 64 Bit. Über den Adreßbus werden alle Adressen vom Prozessor zum Speicher verschickt, um den Platz zu bestimmen, auf den lesend oder schreibend zugegriffen werden soll. Dies kann aber auch die Adresse des Ein-/Ausgabebausteins sein, von dem Daten geliefert werden sollen. Der Adreßbus wird also nur in einer Richtung genutzt.
- **Steuerbus**: Unter diesem Begriff werden verschiedenartige Signalleitungen zusammengefaßt, die der Synchronisation und Verständigung des Prozessors mit allen anderen Bausteinen der Prozessorkarte dienen. Beispielsweise signalisiert die R/W–Leitung die Transportrichtung auf dem Datenbus: R (Read) für die Richtung zum Prozessor, W (Write) für die Richtung zum Speicher. Zum Steuerbus zählen auch die Interruptleitungen.

6.3.6 Die Interrupts

Um auch während des Ablaufs eines Programms auf wichtige Ereignisse, wie Eingabe eines Kommandos über die Tastatur, Signale von externen Schnittstellen usw., reagieren zu können, muß es einen Weg geben, damit die externen Karten auf den Steckplätzen, der DMA– und die Ein-/Ausgabebausteine, den Prozessor ansprechen können. Hierfür gibt es die Unterbrechungssignale (engl. *interrupts*). Jede Einheit, die den Prozessor erreichen will, sendet dem Prozessor einen Interrupt, der ihn auffordert, das aktuelle Programm, das er gerade bearbeitet, zu unterbrechen und sich den neuen Anforderungen zu

Das **Diskettenlaufwerk** stellt ein transportables Speichermedium zur Verfügung, die Diskette oder Floppydisk. Eine Diskette ist ein Speichermedium, das im PC–Bereich eine Datenmenge von 360 Kilobyte bis 2,88 Megabyte aufnehmen kann. Die z.Zt. gängige Größe einer Diskette ist 3.5 Zoll mit 1,44 MB Speicherkapazität. Gelegentlich findet man noch alte 5.25 Zoll Disketten. Die Zugriffszeiten liegen im 200 Millisekunden–Bereich. Die Speicherung von Daten erfolgt auf einer dünnen Kunststoffscheibe mit einer magnetisierbaren Oberfläche.

Abbildung 6.4: Diskettenlaufwerk

widmen. Gibt der Prozessor dem statt, rettet er den aktuellen Zustand des vorherigen Programms auf den Stapelspeicher (auch: Keller oder engl. *stack*), arbeitet die Anforderungen ab und fährt danach mit dem vorherigen Programm weiter. Dies geschieht, indem er sich dessen aktuellen Zustand wieder aus dem Speicher holt.

6.4 Peripherie

Ein Rechner nur bestehend aus der Zentraleinheit ist nicht komplett, da er nicht arbeitsfähig ist. Aus dem Vorhandensein von Steckplätzen sehen wir, daß Erweiterungen möglich, manche davon notwendig sind. Diese Erweiterungen, die sich in der Umgebung der CPU befinden, nennt man auch **Peripherie** (-geräte).

Sehen wir uns die einzelnen Peripherieeinheiten genauer an.

6.4.1 Plattenlaufwerke

Unter Plattenlaufwerken allgemein verstehen wir Disketten- und Festplattenlaufwerke.

Das Floppydisklaufwerk hat einen mechanischen Einschub für die Diskette, einem Motor, der die Diskette mit einer festen Geschwindigkeit rotieren läßt und einem Floppydiskcontroler. Dieser Controler — von engl. *to control* = steuern — verwaltet den oder die magnetischen Schreib-/Leseköpfe, die sich im Laufwerk befinden. Diese können Informationen in magnetischer Form auf die Diskette schreiben oder magnetisch abgespeicherte Informationen von der Diskette lesen. Um überhaupt Informationen auf eine Diskette schreiben zu können, muß sie **formatiert** werden. Dieser Vorgang wird von einem Betriebssystemprogramm ausgeführt (siehe Kapitel 3).

Der Floppydiskcontroler wird von einem speziellen Programm gesteuert. Der direkte Datenaustausch beim Lesen oder Schreiben von Daten wird über den DMA–Kanal abgewickelt. Diese Vorgänge bleiben dem Benutzer des Rechners verborgen, da dafür spezielle Programme des Betriebssystems, die Treiber, zur Verfügung stehen und im Betriebssystem benutzerfreundliche Kommandos zur Verfügung stehen.

Die Spuren aller Plattenoberflächen zusammen, die sich mit einer Einstellung des Arms erreichen lassen, nennt man **Zylinder**. Ein Festplattencontroler arbeitet prinzipiell wie ein Floppydiskcontroler. Der Unterschied liegt in den Zugriffszeiten der Festplatte, der Speichergröße und der Übertragungsrate der Daten zum Rechner. Die schnellsten Zugriffszeiten auf die Daten liegen um 10 Millisekunden. Die Speichergrößen einer Festplatte liegen zwischen 100 Megabyte und dem Gigabytebereich. Die Übertragungsraten der Festplatten sind durch neuere Schnittstellenprotokolle (z.B. SCSI) zur CPU gestiegen und liegen im Bereich 100 Kilobyte/s und etwa 10 Megabyte/s.

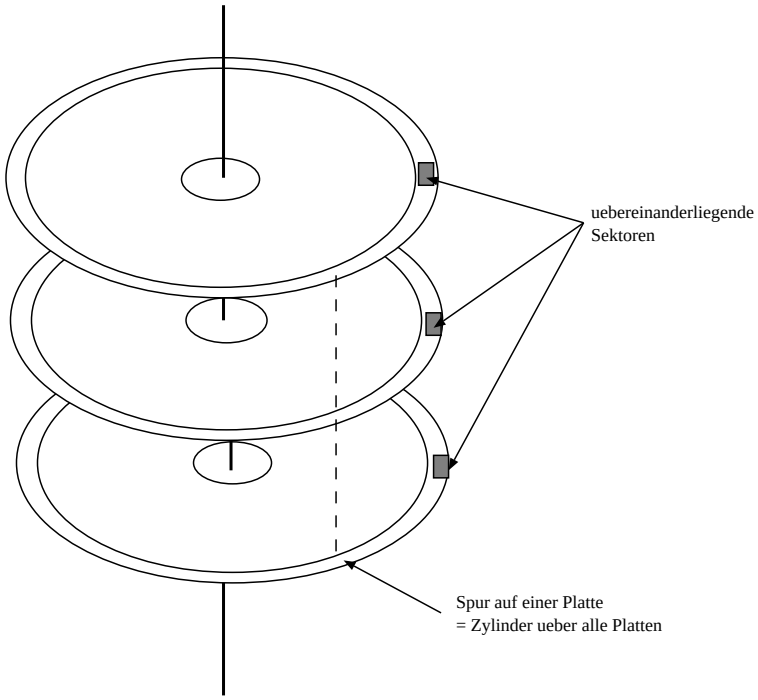


Abbildung 6.5: Darstellung einer Festplatte

Im Unterschied zum Diskettenlaufwerk ist bei einer **Festplatte** das magnetische Speichermedium in einem versiegelten Gehäuse eingebaut („Fest“ heißt hier „nicht auswechselbar“). Festplatten bestehen aus einem Stapel übereinander angeordneter Scheiben, die mit einem Abstand untereinander auf einer gemeinsamen Welle befestigt sind (s. Abb. 6.5). In den Abstand greift der Arm mit den Schreib-/Leseköpfen ein. Für jede Plattenoberfläche gibt es einen Schreib-/Lesekopf. Jede Plattenoberfläche wird durch den Formatiervorgang in Spuren und Sektoren eingeteilt. Alle Köpfe sind an einem horizontal beweglichen Arm so befestigt, daß sie bei jeder Stellung des Arms auf Spuren gleicher Spurnummer auf den verschiedenen Plattenoberflächen zeigen.

Abbildung 6.6: Festplatte

- Textdarstellung mit Position und Größe
- Zeichnen von Punkten, Linien, Rechtecken und Kreisen
- Zoomen von Bildschirmbereichen
- Scrollen (Rollen) des Bildschirms
- Darstellen verschiedener Bildausschnitte
- Lichtgriffeloperationen

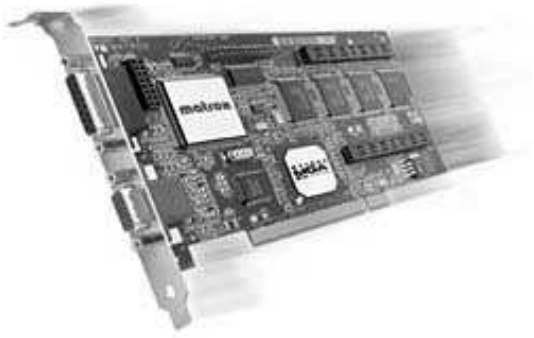


Abbildung 6.7: Grafikkarte

6.4.2 Grafikkarte und Bildschirm

Die **Grafikkarte** dient zur Anzeige, Berechnung und Bewegung von Grafiken und Informationen, die der Rechner auf speziellen Grafikbildschirmen (z.B. bei Workstations, PCs) ausgibt. Man spricht auch von GPU (engl. *Graphic Processing Unit*). Sie enthält dazu oft einen leistungsfähigen extra (Grafik-) Prozessor und benötigt einen eigenen Speicher für die Bildinformationen. Aktuelle Grafikkarten können über den AGP-Bus (*Accelerated Graphics Port*) für speicherintensive Anwendungen auch auf den Hauptspeicher zugreifen.

Die Größe des Speichers steigt rasch mit den Anforderungen an Auflösung (Anzahl der Bildpunkte (engl. *pixel*) und Farbqualität (Anzahl der Farben). Die Berechnung der notwendigen Speichergröße ist relativ einfach: $\boxed{\text{Auflösung} \cdot \text{Farbtiefe}}$

Hat man also eine Bildschirmauflösung von 1024×768 Pixel und 65536 (2^{16}) Farben gewählt, benötigt man: $1024 \cdot 768 \cdot 2 = 1572864\ Byte = 1,5MB$ Speicher. In diesem Fall werden 16 Bit pro Pixel für Farbinformationen bereitgestellt.

Die Grafikkarte leistet die Darstellung von Grafiken und die Darstellung von Textzeichen. Die Darstellung von Grafiken erfordert Befehle zum Zeichnen eines Punktes, einer Linie, eines Kreises, eines Rechtecks, sowie zum Füllen von Flächen und zum Bewegen von ausgewählten Regionen über den Bildschirm. Sogenannte intelligente Grafikkarten verfügen über Grafikprozessoren, bei denen solche Befehle direkt zum Befehlssatz gehören:

Diese Befehlsvielfalt, gekoppelt mit einer hohen Verarbeitungsleistung, ermöglicht die Darstellung anspruchsvoller Grafiken in allen Anwendungsbereichen. Einfachere Grafikkarten ermöglichen lediglich das Setzen und Löschen eines Pixels. Bei diesen Karten müssen auf der CPU des Rechners ablaufende Programme (Treiber) dafür sorgen, daß z.B. der Befehl „Fläche füllen“ in eine Folge von Pixel–setz–befehlen umgesetzt wird. Die Darstellung von Text auf dem Bildschirm erscheint hier extra, weil die Grafikkarten auch in einem Textmodus arbeiten müssen. Die Darstellung von Text in Grafiken ist ebenfalls sehr oft erforderlich. Für die Darstellung muß die Grafikkarte den Monitor mit den erforderlichen Signalen versorgen. Es gibt Monitore, die mit analogen Signalen (RGB–Signal, Red–Green–Blue oder Video–Signal) und digitalen Signalen (z.B. TTL–Signal: Digitalimpulse mit festgelegtem Spannungswert (Pegel)) arbeiten. Multisync–Monitore können sich selbsttätig an verschiedenartige Signale anpassen.

Für Grafikkarten existieren sehr viele Standards, die sich auf die Grafikbefehle beziehen (ANSI–Normen, wie X3LH3.6 CG-VDI für GEM, X11 und andere Benutzeroberflächen) oder nur auf die Schnittstelle der Karte nach außen (im PC–Bereich z.B. VGA, SVGA). Um Grafikkarten in diesen Standards zu betreiben, benötigt man einerseits Treiber, die die Karte aus dem Betriebssystem heraus bedienen, und zum anderen Softwarepakete, die die Entwicklung eigener Software oder Programme ermöglichen.

6.4.3 Ausgabegeräte und Kommunikation

Ein wichtiger Aspekt der Mensch–Computer–Kommunikation ist die **Ausgabe** der vom Rechner erzeugten Ergebnisse. Für die Ausgabe haben wir bereits den Bildschirm kennengelernt. Meistens wollen wir jedoch die Information (Text, Grafik) aus unterschiedlichen Gründen auf Papier ausgeben (Stichwort für die Zukunft: „Papierloses Büro“). Für die Textausgabe werden Drucker unterschiedlichster Art verwendet.

Der Anschluß dieser und anderer Geräte geschieht über sogenannte Schnittstellen (engl. *interface*). Das sind elektronische Baugruppen, die entweder als Erweiterung des Rechners in einem der freien Steckplätze (vgl. Abbildung 6.2) eingebaut werden oder in die Hauptplatine integriert sind, und die die Kommunikation des Rechners mit dem Ausgabegerät nach festgelegten Regeln (Kommunikationsprotokoll) bewerkstelligen. Wir werden Schnittstellen im Kapitel 8 genauer behandeln.

Drucker (engl. *printer*) sind Geräte zur Ausgabe von Daten und Programmen auf Papier. Unterschiedliche Anforderungen an die Geschwindigkeit, Druckqualität und Zeichensatz haben zu einer Reihe verschiedener Druckertypen geführt. Nach der Druckertechnik unterscheidet man:

- mechanische Drucker (engl. *impact printer*), die das Papier mechanisch unmittelbar berühren
  - Schreibraddrucker
  - Kugelkopfdrucker
  - Nadeldrucker
  - Kettendrucker
- nichtmechanische (engl. *nonimpact printer*), die den Druck ohne direkte mechanische Papierberührung erstellen.
  - Laserdrucker
  - Tintenstrahldrucker

Schreibrad-, Kugelkopf- und Kettendrucker gehören zu den **Typendruckern**, bei denen für jedes druckbare Zeichen ein Typenträger vorgesehen ist. Das Prinzip dürfte von der Schreibmaschine her bekannt sein. Diese Druckerarten sind heute nur noch von marginaler Bedeutung.

Zu den **Matrixdruckern** zählen Nadel- und Tintenstrahldrucker. Bei ihnen wird jedes Zeichen aus einzelnen Punkten einer Matrix zusammengesetzt (s. Abbildung 6.8).

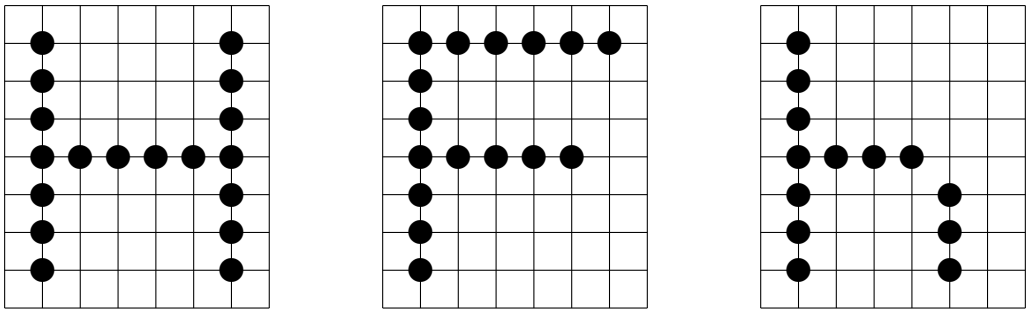


Abbildung 6.8: Zeichendarstellung in einer Punktmatrix

Werden die Zeichen durch einzelne Nadeln erzeugt, die entsprechend dem auszudruckenden Zeichen aus dem Druckkopf hervorschnellen und das Farbband an das Papier drücken, so spricht man von **Nadeldruckern**.

Tintenstrahldrucker arbeiten sehr leise, können (wie auch Nadeldrucker) als Mehrfarbdrucker ausgebildet sein und haben im Vergleich zu Nadeldruckern oft ein besseres Schriftbild. Die Druckgeschwindigkeit von Matrixdruckern liegt je nach Druckqualität zwischen 30 und 800 Zeichen. Aufgrund der vielen Variationsmöglichkeiten des Punktrasters können mit Matrixdruckern viele verschiedene Schriftarten sowie Symbole und Sonderzeichen dargestellt werden. Matrixdrucker sind in der Regel grafikfähig. Bei den bisher genannten Druckern handelt es sich um Zeilendrucker, die eine Seite zeilenweise aufbauen.

Abbildung 6.9: Tintenstrahldrucker

Laserdrucker gehören zu den Seitendruckern, die die gesamte Seite auf einmal drucken. Die durch einen Laserstrahl erzeugten Zeichen werden elektrophotografisch auf das Papier gebracht. Ihre Funktionsweise ist vergleichbar mit der eines Fotokopiergerätes. Die Druckgeschwindigkeit beträgt bei Tischdruckern 6 – 10 Seiten/min. Wegen der freien Bewegungsmöglichkeiten des Laserstrahls können Laserdrucker sowohl beliebige Zeichen ausgeben als auch grafische Darstellungen zeichnen. Der Nachteil eines Laserdruckers ist (noch) sein höherer Preis, Vorteile sind hohe Druckgeschwindigkeit und hervorragende Druckqualität. (Die Druckvorlage für dieses Manuskript wurde auf einem Laserdrucker gedruckt.)

Abbildung 6.10: Laserdrucker

Bei **Tintenstrahldruckern** entfällt das Farbband. Stattdessen sind die Punkte der Matrix als feine Röhrchen ausgebildet, durch die je nach zu druckendem Zeichen Tinte auf das Papier gespritzt wird.

Ein **Plotter** (engl. *to plot* = zeichnen) ist ein elektromechanisches Ausgabegerät zum Zeichnen von Kurven. Ein Flachbettplotter verfügt über einen oder mehrere (auch verschiedenfarbige) Zeichenstifte, die an einem Schlitten befestigt sind. Der Schlitten kann durch Motoren in zwei senkrecht zueinander verlaufenden Richtungen, die x– und y–Richtung, bewegt werden. Bei Walzenplottern ist der Zeichenschlitten nur in der y–Richtung beweglich. Das Zeichnen in x–Richtung wird dann durch die Bewegung des Zeichenpapiers geleistet.

Unter **Kommunikation** verstehen wir hier den Daten- und Informationsaustausch mit anderen Rechnern. Der andere Rechner kann Ressourcen und Dienste zur Verfügung stellen wie Rechenleistung oder Speicherplatz (Fileserver in Netzwerken). Diese Kommunikation wird ebenfalls über die im Rechner eingebauten Mehrzweckschnittstellen (siehe Kapitel 8) oder über besondere Netzwerkschnittstellen (siehe Kapitel 9) abgewickelt. Die Überwachung des Kommunikationsprotokolls geschieht durch die Schnittstellenkarte sowie durch auf dem Rechner laufende Treiberprogramme.

Ausführliches zur Kommunikation in offenen Netzwerken (offene Netze: grundsätzlich sind alle Arten von Stationen für unterschiedlichste Kommunikationsdienste anschließbar) folgt in Kapitel 9.

Ein Beispiel für die Rechnerkommunikation ist die Verbindung zweier Rechner über das Telefonnetz. Zwischen Rechner und Telefonnetz wird jeweils ein **Modem** (**Mo**dulator **Dem**odulator) geschaltet. Das ist ein Gerät zur Wandlung digitaler Signale in analoge Signale und umgekehrt. Man kann sich das Modem vorstellen als Telefonapparat, dessen Funktion Wählen, Hören und Sprechen direkt vom Computer angesteuert werden.

Abbildung 6.11: Modem

Abbildung 6.12: ISDN–Karte

Heute sind schon sehr viele Telefonnetze digitalisiert. Es ist also nicht notwendig digitale Signale erst in analoge umzusetzen. Ein Beispiel dafür ist ISDN — engl. **i***ntegrated* **s***ervices* **d***igital* **n***etwork*. Über entsprechende **ISDN–Karten** kann ein Computer direkt an das Netz angeschlossen werden.

6.4.4 Eingabegeräte

Als wichtigstes Eingabegerät haben wir bereits die Tastatur kennengelernt. Daneben gibt es noch andere alternative Eingabemöglichkeiten, z.B. Maus, Lichtgriffel, Grafik–Tablett, Trackball, Joystick, Touch–Screen, und (noch in der Entwicklung begriffen) Spracheingabe. Auch Geräte zur Eingabe von Texten und Grafiken, wie Klarschriftleser und Scanner müssen erwähnt werden.

Während die Spracheingabe die Tastatur vollständig ersetzen kann, dienen die anderen Geräte der ersten Gruppe lediglich dazu, die Bedienung des Computers benutzungsfreundlicher und tastaturunabhängiger zu gestalten. Das wichtigste Gerät aus dieser Gruppe ist die **Maus** (engl. *mouse*). Die Maus, deren zentrales Element meist eine Rollkugel ist, wird auf dem Tisch hin und her bewegt, um den Cursor oder ein anderes Markierungssymbol auf dem Bildschirm zu steuern. Mit zusätzlichen Maustasten können angefahrene Positionen markiert werden. Die Maus wird häufig mit menügesteuerten Programmen eingesetzt, wobei angezeigte Kommandos zunächst mit der Maus angewählt werden und anschließend durch Tastendruck ausgeführt werden (anklicken).

Der **Trackball** ist im wesentlichen eine Maus, deren Rollkugel sich auf der Oberseite befindet und dort direkt mit der Hand bewegt wird. Der Trackball kann daher ortsfest betrieben werden.

Das **Grafik–Tablett** besteht aus einer rechteckigen Fläche, die über zahlreiche Sensoren verfügt und auf der mit einem stiftähnlichen Werkzeug (**Stylus**) oder mit einer **Digitalisierlupe** (ähnlich einer Maus ohne Rollkugel) Konturen gezogen werden können. Dabei werden die Sensoren angesprochen, deren Impulse auf dem Bildschirm dargestellt werden. Das auf das Tablett eingegebene Bild erscheint also auf dem Bildschirm zur Kontrolle. Zusammen mit den Funktionstasten des Zeigegerätes kann das Grafik–Tablett auch statt einer Maus eingesetzt werden.

Der **Lichtgriffel** (engl. *light pen*) ist ein Zusatzgerät, das direkt mit dem Bildschirm arbeitet. Es ist ein Stift, in dessen Spitze anstelle einer Mine eine Photozelle eingebaut ist, die an den Computer angeschlossen ist. Durch Berühren der Bildschirmfläche mit der Griffelspitze können Punkte und Linien gezeichnet oder Bildelemente markiert werden. Die Stiftposition stellt der Computer so fest: mit einer für das menschliche Auge nicht wahrnehmbaren Geschwindigkeit wird der Reihe nach jeder Punkt des Bildschirms einmal

hell und dunkel geschaltet. Der Punkt, dessen Hell–Dunkelschaltung in der Photozelle einen Stromimpuls auslöst, entspricht der augenblicklichen Stiftposition.

Der **Joystick** funktioniert ähnlich dem Steuerknüppel eines Flugzeugs. Besonders bei Computerspielen wird er zur Bewegung und Veränderung von Darstellungselementen der Bildschirmfläche verwendet. Er besitzt einen oder mehrere Knöpfe zur Funktionsauslösung.

Ein **Scanner** gestattet die Eingabe beliebiger Vorlagen (Text und Grafik) in den Rechner. Hierbei wird die Vorlage in einem sehr engen Raster punktweise abgetastet (gescannt) und als Pixeldatei rein punktgrafisch im Rechner gespeichert und weiterverarbeitet. Auf diese Weise können z.B. Grafiken in mit Textverarbeitungsprogrammen erstellte Manuskripte integriert werden.

Klarschriftleser arbeiten im Hardwareteil wie ein Scanner. Mit ihrer Hilfe werden Schreibmaschinenschrift, Druckschrift und auch deutliche handgeschriebene Zeichen erkannt. Das eingelesene Punktraster wird durch ein spezielles Programm mit vorgegebenen Mustern verglichen und in eine Zeichenfolge umgewandelt. Durch die Tabelle der vorgegebenen Musterzeichen kann man die Fehlerquote bei der Zeichenerkennung einheitlich geschriebener Texte unter 1 Prozent, bei besonders sauberen Vorlagen auch unter 1 Promille drücken.

Vergleichbar mit dem Problem der Zeichenerkennung ist das Problem der Spracherkennung, was sich bei Geräten zur **Spracheingabe** stellt. Diese Geräte dienen zur Eingabe von vom Menschen gesprochenen Worten und Sätzen. Neben dem Problem der Aussprache ist die natürliche Sprache vor allem wegen der Unschärfe ihrer Semantik (Semantik: ist in einer Sprache die Beziehungen zwischen den Sprachzeichen oder Worten und den Bedeutungen, die sie haben. Ausdrücke in natürlichen Sprachen können im allgemeinen vieldeutig sein) für den Rechner nur mit bestimmten, sehr aufwendigen Hilfsmethoden erkennbar und verarbeitbar. Daher sind Spracheingabegeräte noch im Entwicklungsstadium.

6.4.5 Speichermedien

Als Speichermedien haben wir bereits den Haupt- oder Arbeitsspeicher des Rechners und Plattenlaufwerk kennengelernt. Der Arbeitsspeicher ist wie in Abschnitt 6.3.2 beschrieben als RAM ausgeführt, d.h. Speicher mit wahlfreien Schreib- und Lesezugriff. Bestimmte Grundbestandteile des Betriebssystems (bei PCs das BIOS — engl.**b***asic* **i***nput/***o***utput* **s***ystem*) sind oft im ROM gespeichert. Der Inhalt des ROMs wird bereits bei der Herstellung festgelegt und kann nicht mehr verändert werden. Spezielle Ausführungen von ROMs wie **EPROM** (engl. **e***rasable* **p***rogrammable ROM* — lösch- und programmierbarer ROM) erlauben das Löschen des Inhalts durch z.B. UV–Strahlung und sein Neuprogrammieren.

Der Arbeitsspeicher des Rechners kann auf der Hauptplatine oder durch Einbau von **Speichererweiterungskarten** vergrößert werden. Diese Speichererweiterungen werden in freie Steckplätze eingebaut und enthalten neben den RAM–Bausteinen noch Elektronikbausteine zur logischen Anbindung des Speichers an den Prozessorbus. Durch die hohe Ausbaufähigkeit heutiger Mainboards dürfte man Speichererweiterungskarten nur noch in älteren Rechnern antreffen.

Speichermedien zur längerfristigen Datenhaltung sind neben den angeführten Magnetplatten z.B. Magnetbänder oder optische Platten.

Demgegenüber sind **Magnetbänder** und **-kassetten** Speicher mit sequentiellem Zugriff, da auf Informationen an einer bestimmten Stelle erst dann zugegriffen werden kann, wenn die davorliegenden Informationen gelesen worden sind. Magnetbänder speichern Daten genau wie von Tonbändern oder Compact–Kassetten her bekannt durch Ummagnetisierung der magnetischen Schicht des Bandes.

6.4.6 Zusätzliche Hardware

In diesem Abschnitt soll etwas über spezialisierte Hardware gesagt werden.

Optische Platten (CD — engl. *compact disc*) werden mit Lasertechnik sowohl beschrieben als auch abgetastet. Beim Beschreiben wird die digitale Information binär in die Platte eingebrannt (bei CDs für Tonaufzeichnung wird die Information als Folge von kleinen Vertiefungen in die Oberfläche eingepresst). Neuerdings gibt es auch wiederbeschreibbare magneto–optische Platten. Die Aufzeichnungsdichte der optischen Platten ist so groß, daß bereits Kapazitäten von mehreren Gigabyte erreicht werden. Damit eignen sie sich besonders zur langfristigen Speicherung extrem großer Datenmengen. Optische Platten gehören zu den Speichern mit direktem wahlfreiem Zugriff.

Abbildung 6.13: CD–ROM

6.4.6.1 Koprozessoren

Koprozessoren sind intelligente Bausteine oder Baugruppen, die parallel zum Hauptprozessor arbeiten oder auch diesen ersetzen. Sie können für sehr spezielle Aufgaben gebaut sein. Nach ihrer Funktionsweise lassen sich unterscheiden:

- Koprozessoren, die für einen bestimmten Hauptprozessortyp geschaffen wurden und daher auch mit speziellen Signalen ausgestattet sind; sie horchen sozusagen seinen Busverkehr ab und filtern die für sie bestimmten Befehle heraus.

- Koprozessoren, die wie Peripheriebausteine angesprochen werden; sie sind über den normalen Daten- und Adreßbus und über die Funktionscode–Leitungen ansprechbar.

Vom ersten Typ sind z.B. die in den Personal Computern mit Prozessoren vom Typ Intel $80x86$ einbaubaren Numerik–Koprozessoren (FPU — engl. **f***loating* **p***oint* **u***nit*) $80x87$. Diese Koprozessoren sind ausgelegt für arithmetische Operationen im Gleitkommaformat (REAL–Zahlen) und für die Wurzelberechnung, Winkelfunktionen mit ihren Inversen, logarithmische und Exponentialfunktionen. Ab der Prozessorgeneration 80486 ist ein Koprozessor im Hauptprozessor integriert.

Die Verwendung solcher Koprozessoren bringt bei rechenintensiven Programmen Beschleunigungen mit Faktoren in der Größenordnung von 10. Durch die Koprozessorschnittstelle im Hauptprozessor ist es möglich, daß die CPU parallel zur FPU die Arbeit fortsetzt, bis das Ergebnis des Koprozessors benötigt wird.

Koprozessoren vom zweiten Typ sind beispielsweise **Signalprozessoren**, die analog/digital– oder digital/analog–Wandlung von Daten durchführen. Ihr Einsatz wird vom Hauptprozessor gesteuert, oder sie arbeiten sogar unabhängig vom ihm. In diese Klasse fallen auch die in Abschnitt 6.4.2 besprochenen Grafikprozessoren (GPUs).

6.4.6.2 Soundkarten

Soundkarten dienen in erster Linie der Tonwiedergabe und dem Aufzeichnen von Audio–Signalen, beispielsweise Musikstücken oder Sprache. Sie verfügen für diese Aufgaben über diverse Schnittstellen, z.B. Anschlüsse für Mikrofon, CD–ROM, HIFI–Anlagen, Kopfhörer und Lautsprecherboxen. Desweiteren enthalten Soundkarten einen internen Synthesizer, der über MIDI–Daten (engl. **m***usical* **i***nstrument* **d***igital* **i***nterface*: Standard zur Übertragung von Daten elektronischer Musikinstrumente) angesprochen werden kann.

6.4.6.3 Videokarten

Video–Overlay–Karten wandeln ein Videosignal in ein für den Computermonitor verträgliches RGB–Signal um und speisen dies an CPU und Grafikkarte vorbei in den Monitor ein. Sie besitzen eine Eingangsbuchse für das Videosignal des Videorecorders, der Videokamera oder des Fernsehers. Über eine zweite Buchse wird das Signal der Grafikkarte eingespeist, so daß der Monitor direkt an der Videokarte angeschlossen wird. Da Videokarten auch als Framegrabber arbeiten, d.h. sie können einzelne digitalisierte Videobilder als Bilddateien abspeichern, können die Bilder auch mit Hilfe entsprechender Software weiter bearbeitet werden. Jedoch können sie aufgrund der großen anfallenden Datenmengen meist nur Einzelbilder speichern.
Kombiniert z.B. mit einer Videokamera lassen sich Videokarten als äußerst vielseitige 3D–Scanner einsetzen.

Sowohl auf der Soundkarte als auch auf der Videokarte müssen die eingespeisten analogen Daten (das Audio- bzw. das Videosignal) in für den Computer verarbeitbare digitale Daten umgewandelt werden (für die Ausgabe die digitalen in die analogen Daten). Hierfür sind auf den Karten sogenannte AD– bzw. DA–Wandler installiert, die im folgenden näher besprochen werden.

6.4.6.4 Analog–Digital–Wandler

Um einen Computer für Meß- und Steuerungsaufgaben einsetzen zu können, muß eine Schnittstelle zwischen der digitalen Welt des Computers und der analogen Welt außerhalb hergestellt werden. Die Umsetzung analoger Meßdaten in digitale Informationen erfolgt durch Analog–Digital–Wandler (engl. *analogue–digital–converter*, kurz ADC oder AD–Wandler). Die umgekehrte Umsetzung digitaler Informationen in analoge Signale erfolgt durch Digital–Analog–Wandler (engl. *digital–analogue–converter*, kurz DAC oder DA–Wandler).

Ein AD–Wandler setzt eine elektrische Spannung in eine Dualzahl um. Da die Stellenzahl dieser Dualzahl bauartbedingt beschränkt ist, erfolgt die Umwandlung diskret, d.h. einem bestimmten Spannungsintervall wird jeweils dieselbe Zahl zugeordnet. Die duale Stellenzahl des Wandlers wird als Auflösung bezeichnet und in Bit angegeben. Gewöhnliche Wandler haben Auflösungen im Bereich von acht bis sechzehn Bit.

In der Praxis haben sich drei verschiedene Umwandlungsverfahren durchgesetzt:

Zählverfahren: Die zu wandelnde Spannung wird mit einer Referenzspannung verglichen. Diese Referenzspannung ist die kleinste vom Wandler auflösbare Spannung. Ist die Meßspannung größer als die Referenzspannung, wird die Referenzspannung verdoppelt und wieder verglichen. Fällt der Test wieder negativ aus, wird die Referenzspannung verdreifacht und wieder verglichen. Dieser Vorgang wird solange fortgesetzt, bis die Meßspannung kleiner ist als die Referenzspannung. Die Zahl der durchgeführten Vergleiche ist proportional zur Meßspannung und ist der gesuchte Wandlerwert.
Dieses Verfahren erfordert nur geringen Schaltungsaufwand, allerdings ist die Wandlungszeit relativ lang und von der angelegten Spannung abhängig (zwischen 1 und 100 ms).

Wägeverfahren: Wandler, die nach diesem Verfahren arbeiten, bestehen aus einem DA–Wandler (s.u.) und einem Komparator, der die am DA–Wandler ausgegebene Spannung mit der angelegten Meßspannung vergleicht. Bei der Umwandlung wird die angelegte Spannung durch die Spannung am DA–Wandler schrittweise approximiert. Die beste Approximation stellt das Ergebnis der Umwandlung dar. Der Approximationsvorgang liefert in jedem Schritt eine Stelle der gesuchten Dualzahl. Sie benötigt daher soviele Schritte wie die Wandlerauflösung in Bit beträgt.
Dieses Verfahren besitzt eine relativ hohe und von der anliegenden Spannung unabhängige Wandlungsgeschwindigkeit.
Wandler dieser Bauart werden z.B. in der digitalen HiFi–Technik eingesetzt.

Parallelverfahren (Flash–Wandler): Die zu wandelnde Spannung wird gleichzeitig mit allen vom Wandler darstellbaren Werten verglichen. Dadurch wird die beste Approximation in einem Schritt

gefunden. Diese hohe Geschwindigkeit wird allerdings mit einem extrem hohen Schaltungsaufwand erkauft. Zum Beispiel werden bei einer Auflösung von nur acht Bit 256 Komparatoren und 256 Referenzspannungen benötigt.
Wandler dieser Art kommen nur dort zum Einsatz, wo es besonders auf hohe Geschwindigkeiten und weniger auf große Auflösung ankommt, z.B. in Digital–Sampling–Oszilloskopen (Wandlungszeit 20 ns und weniger) oder in der digitalen Echtzeit–Videobildverarbeitung.

Bei allen Verfahren darf sich das Eingangssignal während der Wandlungszeit nicht ändern, da ansonsten falsche Ergebnisse geliefert werden würden. Zu diesem Zweck wird vor den Wandler meist eine sog. Sample–and–Hold–Schaltung gesetzt. Diese Schaltung „friert“ den Wert des Eingangssignals für einen Moment ein.

6.4.6.5 Digital–Analog–Wandler

Ein DA–Wandler setzt eine Dualzahl in eine elektrische Spannung um. Auch hier erfolgt die Umwandlung aufgrund der beschränkten Stellenzahl diskret, d.h. die elektrische Spannung kann nur in (kleinen) Stufen geändert werden.

Im Gegensatz zum AD–Wandler basieren DA–Wandler auf einem einheitlichen Grundkonzept, dem Widerstandsnetzwerk. Hierbei ist für jede Stelle der Dualzahl ein Schaltelement vorgesehen, das eine dem Stellenwert entsprechende Spannung ein- oder ausschalten kann. Die einzelnen Teilspannungen werden zu einer Gesamtspannung zusammengefaßt, die der Analogdarstellung der Dualzahl entspricht.

6.4.6.6 Digitale Signalprozessoren (DSPs)

Mit der Verbreitung der digitalen Signalübertragung und -speicherung (z.B. Compact–Disc) entstand der Bedarf, auch die Signalbearbeitung digital durchzuführen. Das bedeutet, daß nicht mehr das analog aufbereitete Signal digitalisiert wird, sondern das Signal sofort umgewandelt wird und alle Bearbeitungsschritte anschließend durch Rechenoperationen mit den digitalen Informationen durchgeführt werden. Praktisch alle analogen Signalbearbeitungsstufen wie z.B. Filter lassen sich mit einem Computer nachbilden. Allerdings ist dafür eine sehr große Rechenleistung erforderlich, denn Signalverarbeitung muß in **Echtzeit** erfolgen. Aus diesem Grund wurden spezielle, digitale Signalprozessoren entwickelt (DSP), die in Verbindung mit einem AD–Wandler (und ggf. einem DA–Wandler) analoge Signalbearbeitungsstufen digital nachbilden können.

Dieses Verfahren hat zwei wesentliche Vorteile. Zum einen besitzt es eine hohe Flexibilität: ein Analogfilter kann im ungünstigsten Fall nur durch Austausch einzelner Bauelemente in seiner Charakteristik verändert werden, also durch Änderung der Hardware. Dagegen genügt bei einem digitalen Filter eine Änderung der Software. Zum zweiten ist die digitale Signalbearbeitung absolut stör- und rauschfrei, bei analogen Verfahren ist jedes verwendete Bauelement eine Quelle für Rauschen und andere Störungen.

Neben der Nachbildung analoger Filter lassen sich mit DSPs auch andere Operationen durchführen, die in Analogtechnik kaum oder gar nicht möglich sind. Ein Beispiel hierfür ist die Echtzeit–Spektralanalyse von Signalen mit Hilfe der (Fast–)Fourier–Transformation.

Die moderne HiFi–Technik ist ein Beispiel dafür, wie weit die digitale Signalverarbeitung heute verbreitet ist. Auf Compact–Discs ist immer häufiger das DDD–Kürzel zu finden, das bedeutet, daß bei der Aufnahme im Studio das Mikrofonsignal sofort digitalisiert wurde, und daß alle weiteren Bearbeitungsschritte von der Aufzeichnung der Master–Tapes und der Abmischung bis zur Speicherung auf CD rein digital erfolgten.

6.5 Zusammenfassung

Für unterschiedliche Anwendungsbereiche haben sich Kategorien von Rechnern herausgebildet. Sehr lebhaft entwickeln sich Personal Computer, die zunehmend die Qualität von Workstations haben.

Ein Rechner–Arbeitsplatz hat eine Tastatur mit Maus zur Eingabe und einen Bildschirm zur Ausgabe. Das Rechnergehäuse enthält eine Grundplatine, Karten in Einschubplätzen, Festplatten- und Diskettenlaufwerke Alle Komponenten sind durch den Systembus verbunden.

Die Prozessorkarte enthält vor allem den Prozessor und den Hauptspeicher, die durch den Prozessorbus verbunden sind. Ein Taktgenerator erzeugt die nötige Frequenz für Prozessor und Bus.

Im Prozessor lassen sich Datenwerk, Adreßwerk, Steuerwerk und Bustreiber als funktionale Gruppen unterscheiden. Zu jeder funktionalen Gruppe gehören Register, das sind schnelle prozessorinterne Speicherplätze verschiedener Breite.

Der Arbeitsspeicher kann aus Hauptspeicher und Erweiterungsspeicher bestehen. Er dient zum Ablegen und Wiederabrufen von Befehlen und Daten. Die Speicherzellen, Bytes, sind über ihre Adresse durch wahlfreien Zugriff (random access) erreichbar. Dynamisches RAM wird bevorzugt als Speicherbaustein eingesetzt.

Der Prozessorbus läßt sich aufteilen in Datenbus, Adreßbus, Steuerbus. Die Leistungsfähigkeit eines Busses hängt wesentlich von Busbreite und Taktfrequenz ab.

Schnittstellen dienen zum Anschluß von Geräten. Neben der Tastaturschnittstelle ist besonders wichtig der Anschluß von Diskette und Festplatte mittels DMA (engl. *direct memory access*).

Externe Geräte (Peripheriegeräte) werden ebenfalls über Schnittstellen angeschlossen. Die Grafikkarte erzeugt das Bild zur Ausgabe auf dem Monitor. Sie braucht eigenen Speicher für die heute verlangte hohe Auflösung und reiche Auswahl an Farben. Ein eigener Grafikprozessor erlaubt Befehle zur Darstellung und Manipulation grafischer Elemente.

Die Diskette ist ein portables Speichermedium mit Kapazitäten bis zu 2,88 MByte und Zugriffszeiten um 200 ms. Festplatten bieten Platz im Gigabytebereich bei Zugriffszeiten um 10 ms.

Ausgabegeräte ermöglichen die dauerhafte Ausgabe von Text und Grafik auf Papier. Unterschiedliche Anforderungen haben zu verschiedenen Druckertypen geführt. Von den Zeilendruckern bieten Typendrucker mit unveränderbarem Zeichensatz hohe Ausgabequalität oder große Druckleistung. Für Text- und Grafikausgabe durchgesetzt haben sich Matrixdrucker wie die Tintenstrahldrucker, bei denen beliebige Zeichen aus einer Punktmatrix zusammengesetzt werden können. Zu den Seitendruckern, die die gesamte Seite vor dem Druck erzeugen, gehören Laserdrucker, die sich wegen hoher Druckqualität und Leistung rasch weiter durchsetzen werden. Plotter arbeiten wie x–y–Schreiber und dienen der Ausgabe von mehrfarbiger Liniengrafik.

Für den Datenaustausch von Rechnern untereinander über das Telefonnetz gibt es das Modem und ISDN.

Wichtigste Eingabegeräte neben der Tastatur sind die Maus und das Digitalisiertablett, bei denen die Position des Zeigegerätes an den Rechner übertragen wird. Scanner und Klarschriftleser ermöglichen die Eingabe von Text und Grafik ohne Tastatur. Die Spracheingabe wird zukünftig Bedeutung gewinnen.

Wichtige portable Speichermedien sind Magnetbänder, die große Datenmengen sequentiell speichern. Große Zukunft haben (magneto-)optische Platten für extrem große Datenmengen (auch wiederbeschreibbar) mit wahlfreiem Zugriff.

Numerik–Koprozessoren sind spezialisierte Hardware, die abgestimmt auf den Prozessor, REAL–Arithmetik und Funktionsberechnungen extrem schnell durchführen können. [1]

[1] Teile dieses Kapitels basieren auf dem Skript von Prof. Biedl, FB 13.

©W.W. Baumann, Juni 1994, überarbeitet und aktualisiert von Chr. Böhning, Februar 1998

6.6 Fragen und Übungen zu Kapitel 6

F 6.1 Was enthalten unsere PC's ?

F 6.2 Was bedeutet „Bus“ in der Informationstechnik ?

F 6.3 Über welche Busse steht die CPU mit der Außenwelt in Verbindung ?

F 6.4 Wozu ist ein numerischer Coprozessor gut ?

F 6.5 Ordne die folgenden Speicher nach ihrer Schnelligkeit: Hauptspeicher, Floppy–Disk, Festplatte, Prozessorregister

F 6.6 Wer oder was sind Interrupts ?

F 6.7 Welche Abmessungen sind bei Floppy–Disks im PC–Bereich üblich ?

F 6.8 Welche Arten von Registern hat ein Mikroprozessor?

F 6.9 Wieviel Speicher benötigt eine Grafikkarte zur Darstellung eines Bildes mit 800×600 Pixeln bei 256 Farben?

F 6.10 Vervollständige folgende Matrix.

Drucker	Matrix	Type	impact	non–impact	Farbe	monochr.	Zeile	Seite
Kugelkopf								
Nadel								
Ketten								
Laser								
Tintenstrahl								

F 6.11 Was ist der Unterschied zwischen Akkustikkoppler und Modem ?

F 6.12 Welche Eingabegeräte sind in unserem Mikropool vorhanden ?

F 6.13 Was bedeuten die folgenden Abkürzungen : RAM, ROM, EPROM, CD-ROM. Erläutere die Begriffe.

F 6.14 Was sind die Vor- und Nachteile von Magnetbändern ?

6.7 Antworten zu den Fragen und Übungen zu Kapitel 6

A 6.1
- Motherboard
- Festplatte und Diskettenlaufwerk
- Controler für Laufwerke
- Steckkarten (Speicher, Grafik, Schnittstellen, Maus, Netzwerk)
- Netzteil und Lüfter

A 6.2 Eine Anzahl logisch zusammengehöriger paralleler Leitungen.

A 6.3 Adreß-, Daten- und Steuerbus.

A 6.4 Zur schnellen Bearbeitung von Fließkommaarithmetik und mathematischen Standardfunktionen.

A 6.5 Prozessorregister, Hauptspeicher, Festplatte, Floppy–Disk

A 6.6 Interrupts ermöglichen die Unterbrechung eines Programms, um auf „wichtige“ externe (oder interne) Ereignisse reagieren zu können. Interrupts werden meist von der Hardware ausgelöst. Software–Interrupts, z.B. bei Division durch Null, nennt man Traps.

A 6.7 3.5 Zoll (und früher 5.25 Zoll)

A 6.8 Daten-, Adreß- und Befehlsregister sowie Befehlszähler

A 6.9 ca. 512 kByte.

A 6.10

Drucker	Matrix	Type	impact	non–impact	Farbe	monochr.	Zeile	Seite
Kugelkopf		x	x			x	x	
Nadel	x		x		x		x	
Ketten		x	x			x	x	
Laser	x			x	x			x
Tintenstrahl	x		x		x		x	

A 6.11 Beim A. werden Schallwellen erzeugt, die zunächst vom Telefonapparat in Signale umgewandelt werden. Das Modem greift direkt auf die elektrischen Drähte des Telefonnetzes zu.

A 6.12 Tastatur, Maus

A 6.13

RAM: engl. *random access memory* = Speicher mit wahlfreiem Zugriff, d.h. alle Speicherzellen sind in Prinzip gleichschnell zugreifbar. In das RAM kann beliebig oft geschrieben und gelesen werden.

ROM: engl. *read only memory* = Nur–Lese–Speicher, d.h. in diesen Speicher kann nicht (vom Benutzer) geschrieben und damit auch nicht gelöscht werden.

EPROM: engl. *erasable programmable ROM* = lösch- und programmierbarer ROM. Üblicherweise können diese Speicherbausteine durch eine starke UV–Strahlung gelöscht werden. Die (erneute) Speicherung erfolgt bei relativ hohen Spannungen in sogenannten „EPROM–Brennern“.

CD–ROM: engl. *compact–disc ROM* = auf der Technik der Musik–CD basierende Speicherplatte. Große Datenmengen (Lexika, Handbücher, Telefonbücher, etc.) werden von i.a. kommerziellen Anbietern in computerlesbarer Form angeboten.

A 6.14 Vorteil: hohe Speicherkapazität, billig
Nachteil: nur sequentieller Zugriff

Kapitel 7

Mikroprozessoren

Wir werden in diesem Kapitel etwas näher auf Aufbau und Funktion eines Mikroprozessors eingehen. Dabei orientieren wir uns am Intel–Prozessor 8080. Es ist uns natürlich auch bewußt, daß wir damit nicht die neueste Technologie beschreiben, aber die Entwicklung in diesem Bereich ist so schnell, daß viele Informationen im Laufe eines Semesters veralten würden. Außerdem war der 8080 Grundlage für die meisten heutigen Prozessoren in den weitverbreiteten PCs. Desweiteren erscheint es auch nicht angemessen, in einem Einführungskurs die hochkomplizierte Funktionsweise eines z.Zt. aktuellen Prozessors zu beschreiben. Der 8080 bietet die Möglichkeit, an einem relativ einfachen Beispiel die auch heute noch gültigen grundlegenden Abläufe in einem Mikroprozessor darzustellen. Für weiterführende und detailliertere Informationen über heutige Prozessorfamilien empfehlen wir einen Blick in das Angebot der Hersteller im World Wide Web: www.intel.com, www.amd.com, www.digital.dom/semiconductor, www.motorola.com, www.cyrix.com, etc.

7.1 Allgemeiner Aufbau und Funktion

7.1.1 Struktur

Ein **Mikroprozessor** ist die auf einem oder mehreren integrierten Schaltkreisen (IC — engl. *integrated circuit*) untergebrachte CPU eines Mikrorechners. Der Mikroprozessor besteht im wesentlichen aus dem Rechenwerk als Datenprozessor und dem Steuerwerk zur Befehlsverarbeitung und Koordinierung der einzelnen Funktionen und zeitlichen Abläufe. Die Mehrzahl der modernen Mikroprozessoren sind Ein–Chip–Mikroprozessoren, d.h. alle entscheidenden Elemente sind auf einem Chip untergebracht. Bei 16– und 32–Bit–Mikroprozessoren sind auf dem Chip mitunter auch noch Speicherverwaltung und Cache–Speicher integriert (siehe auch Abschnitt 7.5).

Mikroprozessoren mit fester Wortlänge verarbeiten Bitketten bestimmter Länge (8, 16 oder 32 Bit) als zusammenhängende Einheiten. Zur Zeit dominieren die 32–Bit–Mikroprozessoren. 64–Bit–Mikroprozessoren gewinnen an Bedeutung.

Die Struktur eines Ein–Chip–Mikroprozessors ist in Abb.7.1 dargestellt. Das Blockschaltbild stellt den 8–Bit–Mikroprozessor 8080 der Firma Intel dar, den wir uns im Abschnitt 7.4 näher ansehen wollen. Der 8080 ist der Urvater aller Prozessoren der „IBM–Kompatiblen“ PC's.

Andere Mikroprozessoren, auch mit größerer Wortlänge, unterscheiden sich im Aufbau nicht wesentlich davon. Für uns ist die Kenntnis der Registerstruktur eines Mikroprozessors und seiner Adressiermöglichkeiten wichtig, insbesondere zum maschinennahen Programmieren und zum Verständnis moderner Rechnerarchitekturen und Konzepte (Abschnitt 7.5) .

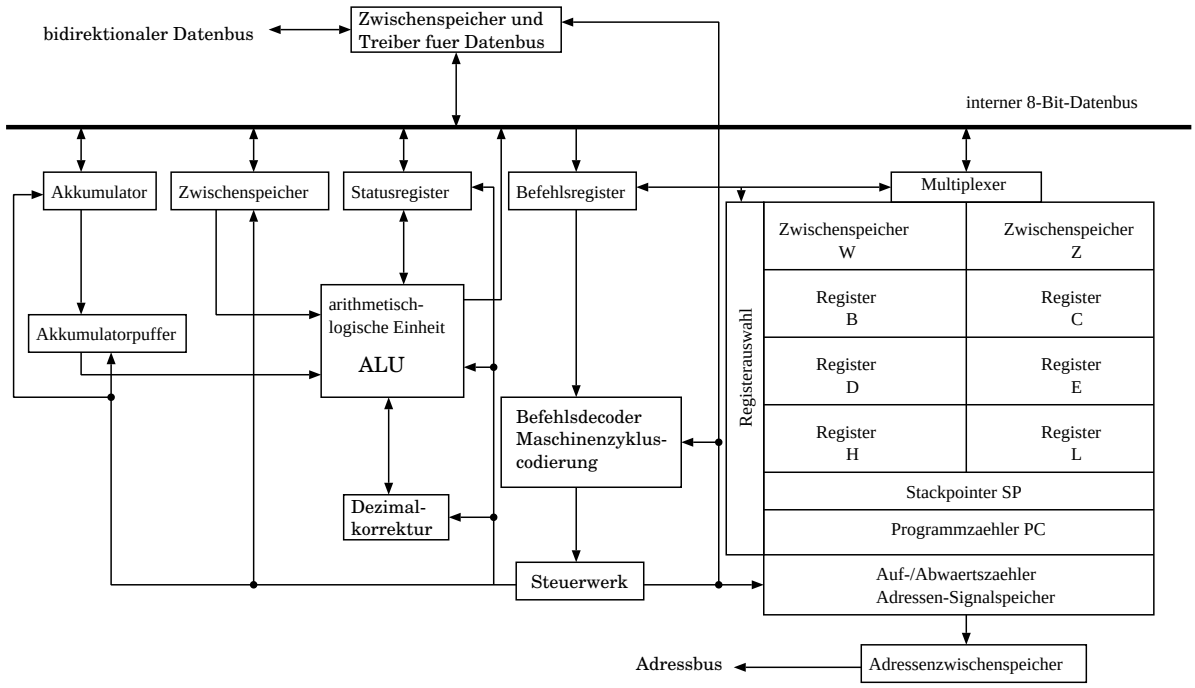


Abbildung 7.1: Blockschaltbild des Intel 8080

Die arithmetisch–logische Einheit führt die arithmetischen und logischen Operationen mit den Operanden aus. Bei der Verknüpfung zweier Operanden befindet sich der eine Operand im Akkumulator und der andere in einem Register des Registerbereichs oder in einer Speicherzelle des Arbeitsspeichers. 8–Bit–Mikroprozessoren besitzen meist einen einzigen Akkumulator oder nur wenige Register, die diese Funktion ausführen können. Bei anderen Mikroprozessoren können oftmals alle Register des internen Registerbereichs auch als Akkumulator wirken. Große Bedeutung für das Programmieren von Verzweigungsoperationen und arithmetischen Berechnungen hat das Status– oder Flag–Register, dessen einzelne Bits in Abhängigkeit vom Ergebnis der ausgeführten Instruktion gesetzt werden (overflow, carry etc.).

7.1.1.1 Register

Der Registerbereich enthält Register, die als Zwischenspeicher für Operanden und Resultate verwendbar sind. Ihre Zugriffszeit ist im allgemeinen wesentlich kleiner als beim Arbeitsspeicher, so daß ihre Verwendung zur Beschleunigung des Programmablaufs und wegen der kurzen Adressen auch zur Speicherplatzeinsparung führen kann. Neben den Datenregistern sind noch Register für spezielle Aufgaben vorhanden.
Hierzu gehören

- Befehls- oder Programmzähler (engl. *program counter*): Der Inhalt verweist auf die Adresse des Arbeitsspeichers, auf die der nächste Zugriff erfolgt.
- Stapelzeiger (engl. *stack pointer*): Der Inhalt gibt die aktuelle Zugriffsadresse des im Arbeitsspeicher stehenden Stack–Speichers (Keller- oder Stapelspeicher, siehe unten) an.
- Indexregister: Der Inhalt wird für indizierte Adressierung genutzt.
- Statusregister: Der Inhalt stellt den aktuellen Zustand des Mikroprozessors dar, z.B. die Bedingungsbits (Flags), Systemzustand und Interruptmaskierung.

Anzahl, Programmiermöglichkeit und Flexibilität der internen Register sind für eine schnelle Programmausführung von erheblicher Bedeutung.

7.1.1.2 Stack

Der Stack (dtsch.: Keller oder Stapel) ist ein Speicherbereich mit gewöhnlich zwei eingeschränkten Zugriffsoperationen. Die eine fügt stets ein Element an das Ende des Stacks an (Einfügen), die andere entfernt stets das letzte Element des Stacks und liefert es als Ergebnis (Ausfügen). Die Einfügeoperation nennt man **push**, die Ausfügeoperation **pop**. Ein Beispiel soll die Funktionsweise eines Stacks verdeutlichen (der Stack sei zu Beginn leer).

Operation	Stack–Inhalt	Ausgabe
-	$< >$	-
push (1)	$< 1 >$	-
push (3)	$< 1, 3 >$	-
pop	$< 1 >$	3
push (2)	$< 1, 2 >$	
push (4)	$< 1, 2, 4 >$	-
pop	$< 1, 2 >$	4
pop	$< 1 >$	2

Das Prinzip, daß stets das zuletzt eingefügte Element eines Speichers als erstes wieder entfernt werden muß, bezeichnet man als LIFO–Prinzip (engl. *last in first out*). Man nennt daher einen Stack auch LIFO–Speicher. Es gibt stackbasierte Programmiersprachen (z.B. Forth) und (Taschen-) Rechner, die mit stackbasierten Operationen arbeiten (UPN — Umgekehrt Polnische Notation, auch Postfix–Notation; erkennbar an der ENTER–Taste statt des Gleichheitszeichens).

7.1.1.3 Steuerung

Das **Steuerwerk** ist für das Lesen der Befehle aus dem Arbeitsspeicher und für deren Interpretation und Ausführung verantwortlich. Das Steuerwerk übernimmt außerdem die Ansteuerung des Arbeitsspeichers und der Ein-/ Ausgabe–Einheiten, wobei über die Anschlußkontakte des Mikroprozessors geeignete Steuersignale ausgetauscht werden. Dekodierung und Ablaufsteuerung des Befehls erfolgen in den meisten Fällen unter Mikroprogrammsteuerung.

Bei der **Mikroprogrammsteuerung** werden die einzelnen Maschinenbefehle des Mikroprozessors, also die im Maschinencode dargestellten Instruktionen des Programms, intern durch eine Folge von elementaren Mikrobefehlen realisiert. Ein Maschinenbefehl entspricht also einer Folge von Mikrobefehlen. Diese Mikroprogramme sind in einem internen ROM abgelegt und dem Anwender nicht zugänglich. Durch die Veränderung der Mikrobefehlsfolge kann der Hersteller gewisse Modifikationen der Befehlsliste des Mikroprozessors durchführen, ohne das Layout des Chips ändern zu müssen.

Daten und **Adressen** werden über Register (Pufferstufen) des Mikroprozessors übertragen. Bei 16– und 32–Bit–Wortlänge werden Daten und Adressen oft zeitlich nacheinander (Zeitmultiplex) über die gleichen Anschlußkontakte ausgegeben. Zur Organisation des Datentransports auf dem Bus des Mikrorechners und zur Kennzeichnung z.B. der sich momentan auf den gemeinsamen Daten- und Adreßleitungen befindlichen Informationen gibt der Mikroprozessor Steuersignale ab. Andere Signale beziehen sich auf die Interruptsteuerung, auf den Prozessorstatus und auf den Zugriff zum Bus.

Tabelle 7.1 gibt eine Übersicht über die Bedeutung der wichtigsten Anschlußkontakte von Mikroprozessoren in verallgemeinerter Form. Für die einzelnen Mikroprozessortypen kann die konkrete Zuordnung

Typ	Funktions-richtung	Verwendung
Adreßleitungen	Ausgabe	oftmals gemeinsame, im Zeitmultiplex genutzte Anschlüsse für Adressen und Daten
Datenleitungen	bidirektional	
Statusleitungen	Ausgabe	aktueller Zustand des Mikroprozessors für Steuerung der externen Logik, z.B. von Adreß- und Datenpuffern
Steuerleitungen	Ausgabe	Steuerung der externen Logik für Lese- und Schreiboperationen und für Ein-/Ausgabe- operationen
Interrupt-anforderungen	Eingabe	maskierbare und nicht-maskierbare Interruptanforderung von externen Einheiten
Prozessorsteuerung	Eingabe	Rücksetzen (reset) in definierten Anfangszu- stand, Wartezyklen, Abschalten vom Bus (Halt- Zustand)
Buszugriffssteuerung	Eingabe, Ausgabe	Busanforderung bei Systemen mit mehreren Busmastern, Quittungssignal für Busübernahme
Taktsignal	Eingabe	meist extern erzeugtes Prozessortaktsignal
Versorgung	Eingabe	meist + 5 V (3,3 V) und Null-Potential

Tabelle 7.1: Allgemeine Anschlußschnittstelle eines Mikroprozessors [10]

stark von der Übersicht in Tabelle 7.1 abweichen. Die Kenntnis des detaillierten Signalspiels des Mikroprozessors mit den angeschlossenen Einheiten ist für den Programmierer nicht erforderlich, ist aber für die Ausarbeitung von Hardware und für das Verständnis ihrer Funktion von entscheidener Bedeutung.

7.1.2 Von–Neumann–Rechner

Die meisten gängigen Mikroprozessoren entsprechen dem Von–Neumann–Prinzip, welches Mitte der Vierziger Jahre von John von Neumann formuliert wurde. Von–Neumann–Rechner sind durch folgende wesentliche Eigenschaften gekennzeichnet:

- Programme und Daten sind im Arbeitsspeicher in gleicher Weise als binär verschlüsselte Informationen (Bitketten) gespeichert. Die Bedeutung dieser Informationen als Instruktionen (Befehle), Daten (Zahlen, Zeichen) oder Adressen (für Speicher, Ein-/Ausgabe) ist nur aus dem Zusammenhang ersichtlich. Programm- und Datenstrukturen sind für die Hardware nicht erkennbar.
- Die Abarbeitung einer Instruktion erfolgt in zwei Phasen, Abb. 7.2. In der ersten Phase wird aus dem Speicherplatz, dessen Adresse im Befehls- oder Programmzähler steht, ein Speicherwort (z.B. 1 Byte oder ein 16–Bit–Wort) gelesen, als Befehl angesehen und im Befehlsdekoder entschlüsselt (**Fetch–Phase**). In der zweiten Phase wird je nach Art des gelesenen Befehls dieser Befehl ausgeführt, oder es werden zunächst weitere, unmittelbar folgende Speicherstellen als Operanden oder Adressen gelesen, ehe die geforderte Operation erfolgen kann. Die Ausführungszeit dieser Ausführungs- oder **Execute–Phase** hängt somit wesentlich von der Befehls- und Adressierungsart ab. Trotz fehlender Identifikation des Speicherinhalts ist ein ordnungsgemäßer Programmablauf möglich, denn es liegt ein zweiphasiger Instruktionsablauf vor, und im Speicher müssen die Operanden und Adressen dem Instruktionscode konsekutiv zugeordnet sein, d.h. der Befehlszähler wird nach jedem Lesen eines Wortes inkrementiert.

Dieser allgemeine Befehlsablauf ist auch für Mikroprozessoren typisch, er bedarf jedoch einiger spezifischer Ergänzungen. Tritt während der Befehlsabarbeitung eine Interruptanforderung auf, so wird diese erst nach vollständiger Abarbeitung des laufenden Befehls bedient. Bei 16– und 32–Bit–Mikroprozessoren können umfangreiche Adreßrechnungen und gegebenenfalls Leseoperationen von Speichersegmenten oder -seiten

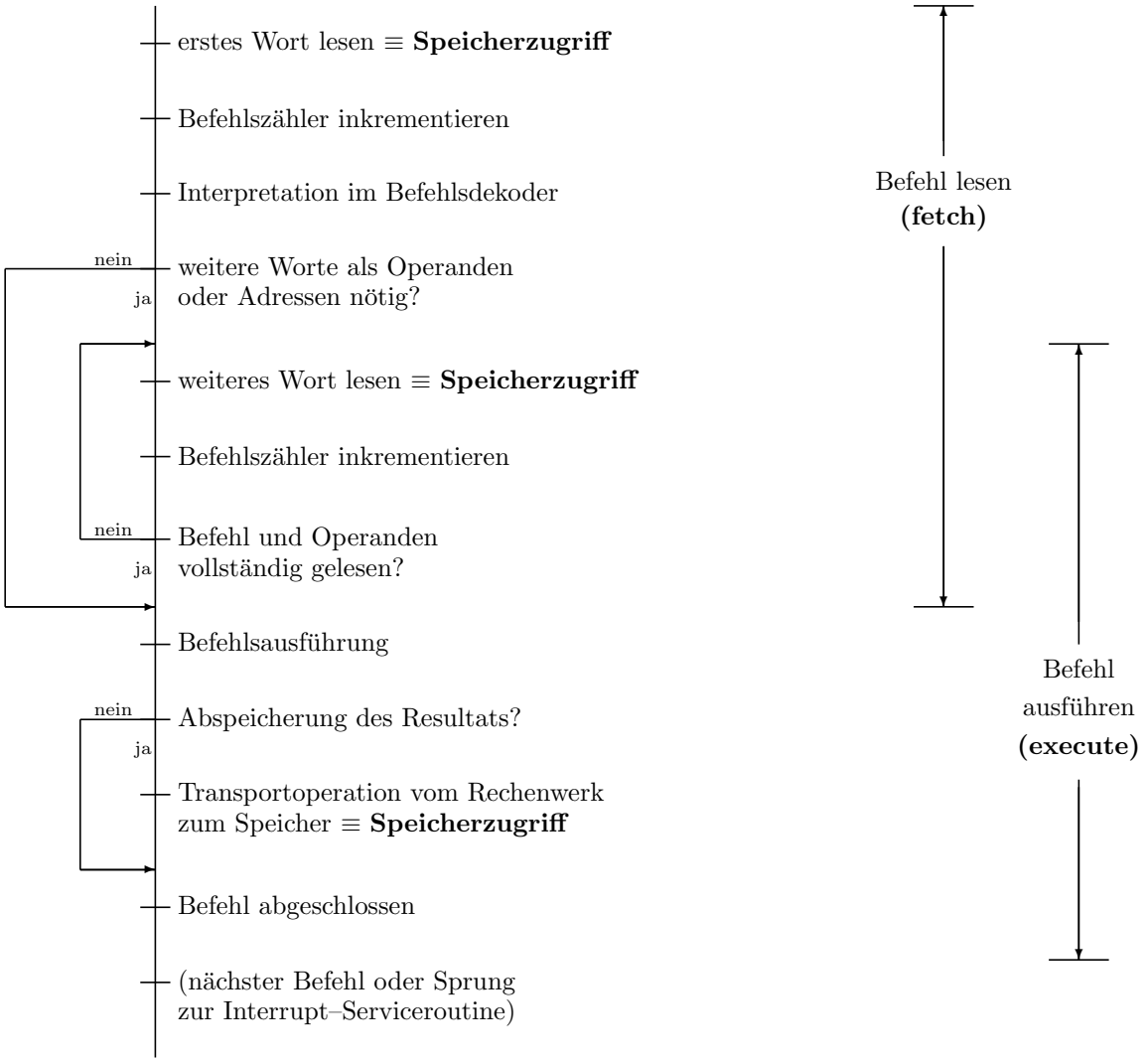


Abbildung 7.2: Befehlsablauf in einem Von–Neumann–Rechner für Mehrwortbefehle ohne Beachtung von Adreßrechnungen und Ein-/Ausgabeoperationen.

aus dem externen Speicher erforderlich werden, bevor die nächste Instruktion des Anwenderprogramms abgearbeitet werden kann.

7.1.3 Operationen

7.1.3.1 Speicherschreib- und Leseoperationen

Der Befehlsablauf für einen Speicherzugriff läßt sich anhand eines Zeitablaufdiagramms verdeutlichen, Abb. 7.3

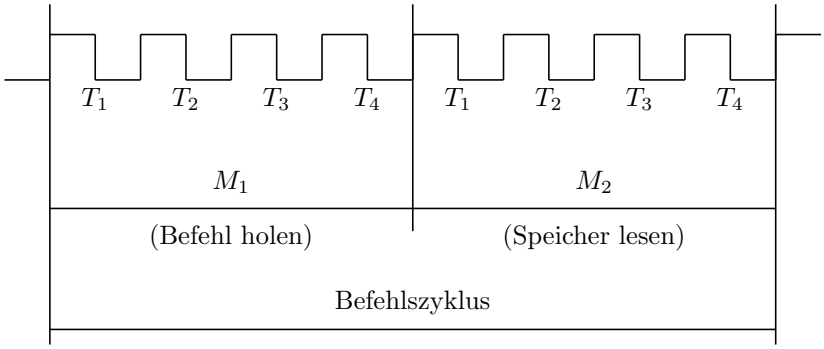


Abbildung 7.3: Befehlszyklus für einen Speicherzugriff (Lesen) [10]

Für das Beispiel eines lesenden Speicherzugriffs lassen sich zwei verschiedene Maschinenzyklen unterscheiden:

- M_1 für Befehl holen (Fetch–Phase)
- M_2 für Speicher lesen

Die Zyklen M bestehen ihrerseits aus Taktzyklen T_1, T_2 usw., die unmittelbar vom Taktgeber des Mikrorechners abgeleitet werden. Zu Beginn des Taktes T_1 tritt der Prozessor in den Zyklus „Befehl holen“ ein. Der Inhalt des Befehls- oder Programmzählers wird als aktuelle Speicheradresse interpretiert und die Speicherzugriffsanforderung (engl. *memory request*) über bestimmte Steuerleitungen als Lesewunsch gemeldet. Ab Takt T_2 wird auf die Antwort des Speichers gewartet. Ist seine Zugriffszeit (engl. *access time*) kürzer als bis zum Ende von Takt T_3 , so werden die Lesedaten zum Zeitpunkt Takt T_3 übernommen. Braucht der Speicher länger, müssen Wartetakte (engl. *waitstates*) eingeschoben werden. Mit diesem Mechanismus findet die Anpassung des Mikroprozessors an Speicherchips mit längerer Zugriffszeit statt. Speicherschreibzugriffe laufen entsprechend ab.

7.1.3.2 Ein-/Ausgabeoperationen

Sind in der Instruktionsliste des Mikroprozessors explizit Ein- und Ausgabebefehle vorhanden, dann steht ein Steuersignal zur Verfügung, das in Verbindung mit der Übertragungsrichtung die Adressierung von Ein- und Ausgabekanälen (Ports) ermöglicht. Der Speicheradreßraum ist damit vom Adreßraum der Ein- und Ausgabeadressen getrennt (isolierte Addressierung).

Von den Kleinrechnern ist die speicherbezogene Adressierung (engl. *memory mapping*) bekannt, die auch bei einigen Mikroprozessoren verwendet wird. Hierbei werden Ein- und Ausgabeadressen wie normale

Speicheradressen betrachtet, können daher auch von allen Befehlen mit Speicherzugriff genutzt werden. Diesem Vorteil steht allerdings gegenüber, daß Speicherbereiche durch feststehende Ein- und Ausgabeadressen blockiert werden können.

7.1.4 Interruptssystem

Eine Interruptanforderung (engl. *interrupt request*, IRQ) ist eine Anforderung nach einer Unterbrechung des laufenden Rechnerprogramms, um eine zeitlich nicht mit diesem Programm korrelierte Interruptserviceroutine (ISR) starten zu können. Diese sogenannten Hardware–Interrupts können z.B. folgende Quellen haben:

- Rechnerinterne Fehler- und Alarmzustände, z.B. bei Zeitüberschreitung (engl. *time–out*) bestimmter Busoperationen, bei Spannungsausfall und bei Fehlern in den Ein-/Ausgabesteuerungen.
- Datenein-/ausgabe angeschlossener peripherer Geräte, wie Dateneingabe nach Tastaturbedienung, Datenanforderung oder Bereitschaftsmeldung eines Druckers oder Anforderung eines direkten Speicherzugriffs (DMA).
- Melde- und Alarmsignale angeschlossener Funktionseinheiten, z.B. Zeitgebersignale einer Echtzeituhr, Meldungen der Prozeßperipherie über unzulässige Anlagenzustände (z.B. zu hohe Temperatur) oder Fertigmeldung eines A/D–Wandlers.

Neben diesen typischen Hardware–Interruptanforderungen gibt es bei Mikroprozessoren noch prozessorintern ausgelöste Software–Interrupts, sogenannte Traps. Sie entsprechen einer Programmunterbrechung nach einem Softwarefehler, beispielsweise bei Erkennen eines nichtdefinierten Operationscodes oder bei Division durch Null. Diese intern ausgelösten Traps treten synchron mit der Befehlsabarbeitung auf.

Eine Interruptanforderung wird wie folgt bearbeitet. Das laufende Programm, das selbst eine Interrupt–Serviceroutine niedriger Priorität sein kann, wird definiert unterbrochen. Dabei wird der aktuelle Prozessorzustand auf den Stack gerettet, damit nach Beendigung der ISR das unterbrochene Programm korrekt fortgesetzt werden kann. Die Interruptquelle wird identifiziert, und es wird zur angeforderten ISR verzweigt. Dabei müssen Prioritäten bei der Bearbeitung mehrerer gleichzeitig vorliegender Interruptanforderungen berücksichtigt und gegebenenfalls zeitweise ausgewählte Interruptanforderungen durch Maskierung unterdrückt werden. Nichtmaskierbare Interrupts führen in jedem Fall zur Interruptbearbeitung.

7.1.5 Bussteuerung

Der Mikrorechnerbus wird in der Regel von der CPU gesteuert. Sie übernimmt bei Datenübertragung die Rolle des Busmasters, während der aufgerufene Speicher oder die adressierte Ein-/Ausgabesteuerung als Slave wirkt. In manchen Fällen kann es im Interesse eines größeren Datendurchsatzes auf dem Bus zweckmäßig sein, den Bus nicht von der CPU steuern zu lassen, z.B. bei direktem Speicherzugriff (DMA) für die Übertragung großer Datenmengen zwischen einer Ein-/Ausgabeeinheit und dem Arbeitsspeicher. Die Möglichkeit, die Bussteuerung anderen Einheiten übergeben zu können, ist besonders bei Multiprozessorsystemen von Bedeutung.

Die Abschaltung des Mikroprozessors vom Bus geschieht über Signale. Bei der einen Methode wird der Mikroprozessor vom DMA–Controler mit einem Haltesignal vom Bus getrennt und quittiert diesen Zustand durch ein Bestätigungssignal (Beispiel: Intel 8080). Nach Rücksetzen des Haltesignals wird der Mikroprozessor wieder zum Busmaster.

Vielseitiger und für den Einsatz in Multiprozessorsystemen geeignet sind Prozessoren mit Busanforderungs- und Busbewilligungssignalen (engl. *bus request* und *bus grant*). Das Signal für die Bestätigung

der Busbewilligung (engl. *bus grant acknowledge*) zeigt an, daß der Bus weiterhin von einer anderen Einheit als Master belegt ist. In dieser Zeit kann der Prozessor durchaus Operationen ausführen, sofern sie sich auf den lokalen Bus beziehen, für den er allein als Master fungiert.

7.1.6 Adressierung

Bei einer Von–Neumann–Architektur erfolgt die Ausführung der Programmbefehle in der Reihenfolge, wie die einzelnen Befehle nacheinander im Speicher abgelegt sind. Bei einem solchen linearen Programm ist daher eine explizite Adressenangabe gar nicht erforderlich. Der Befehlszähler des Prozessors enthält nach Ausführung eines Befehls die Adresse des nachfolgend zu lesenden Befehls. Lediglich bei Sprungoperationen sowie beim Aufruf von Unterprogrammen und Interrupt–Serviceroutinen ist eine Adressenangabe notwendig, um das Sprungziel bzw. die Adresse des ersten Befehls des Unterprogramms zu kennzeichnen.

Die vom Befehl zu verarbeitenden Daten (Operanden) können gemeinsam mit dem Operationscode des Befehls gespeichert sein, in der Regel sind sie jedoch getrennt vom Operationscode im Arbeitsspeicher abgelegt. Daher muß der Befehl die Adresse der Operanden sowie gegebenfalls Angaben darüber enthalten, wie die aktuelle Adresse der Operanden (effektive Adresse) bei der Befehlsausführung berechnet werden kann. Bei den einzelnen Mikroprozessortypen sind zahlreiche Adressierarten implementiert, die sich primär auf die Lokalisierung von Daten beziehen, aber auch für die Ermittlung von Adressen für Sprungoperationen nutzbar sind.

Für die Codierung der Adresse im Befehl und für die Bestimmung der effektiven Adresse gibt es eine Reihe von Adressierungsarten, z.B.:

- indirekte Adressierung
- indizierte Adressierung
- relative Adressierung
- unmittelbare Adressierung

Bei der indirekten Adressierung ist anstelle der Adresse des Operanden, wie bei der direkten Adressierung, die Adresse der Speicherzelle, in der sich die Adresse des Operanden befindet, vorzufinden. Bei der indizierten Adressierung findet eine Adressenverschiebung statt, indem die im Befehl enthaltene Adresse additiv zu einer in einem Register gespeicherten Basisadresse hinzugefügt wird. Bei der relativen Adressierung findet eine indirekte Adressierung in Kombination mit einer indizierten Adressierung, die sich auf die Adresse des Operanden bezieht, statt. Bei der unmittelbaren Adressierung existiert anstelle der Operandenadresse im Befehl eine Konstante, die als Operand gilt.

Befehlsformate Bei Befehlen mit z.B. einem einzigen Operanden muß dessen Adresse als Quelladresse und die Adresse des nach der Befehlsausführung vorliegenden Resultats als Zieladresse angegeben sein. Eine der beiden Adressen kann verdeckt sein, wenn das Resultat in der Speicherzelle des Operanden abgelegt wird und ihn damit überschreibt. Die Adressierung ist „implizit“, wenn bereits aus dem Operationscode die Quell- und Zieladresse bestimmt sind, z.B. bei Operationen mit dem Inhalt des Akkumulators.

Die **Datenformate** des Mikroprozessors unterscheiden sich in der jeweils festgelegten Länge der Bitkette, während eine Kennzeichnung des Datentyps nicht erfolgt (siehe Kapitel 5). Das Datenformat wird eindeutig durch Operationscode und Adressierungsart bestimmt. Die getrennt vom Rechnerprogramm gespeicherten Daten werden meist als Einheiten mit einfacher oder doppelter Wortlänge gespeichert (16 oder 32 Bit).

Die **Adressen**, wie sie vom Programmierer zur Lokalisierung von Operanden u.ä. verwendet werden können, bezeichnet man als **logische Adressen**. Die tatsächlich bei der Befehlsausführung wirksame Adresse, die das Resultat einer Adressberechnung sein kann, stellt die effektive Adresse dar und ist an

den Anschlußkontakten des Mikroprozessors verfügbar. Der logische Adreßraum umfaßt bei einer 16–Bit–Adresse $2^{16} = 64k = 65536$ Speicherplätze. Es gibt Möglichkeiten, diesen Adreßraum zu vergrößern. Den Adreßraum, der vom Arbeitsspeicher genutzt wird, nennt man den Bereich der **physikalischen Adressen**. In älteren Mikrorechnern mit kleinem Arbeitsspeicher überstieg der logische Adreßraum bei z.B. 16–Bit–Adressen oftmals den physikalischen Adreßraum ganz beträchtlich. In neueren Mikrorechnern ist durch billige Speicherchips zwar der physikalische Adreßbereich erheblich angewachsen, andererseits haben virtuelle Speichertechnik und Speicherverwaltungschips (MMU — engl. **m***emory* **m***anagement* **u***nit*) dazugeführt, daß der logische Adreßbereich dem Programmierer praktisch keine Grenzen mehr setzt (siehe hierzu Abschnitt 7.5).

Bei der Ermittlung der effektiven Adresse sind grundsätzlich zwei Stufen zu unterscheiden:

- Adreßberechnung während der Programmübersetzung, d.h. Umrechnung von symbolischen Adressen, Adreßberechnung beim Binden von speicherverschieblichen Objektmodulen, u.ä.
- Adreßberechnung während der Programmausführung, z.B. indirekter und indizierter Zugriff auf Feldelemente, Ermittlung von Zieladressen bei bedingten Sprungoperationen.

Für viele Anwendungen reicht der mit 16 Bit adressierbare Speicherbereich von 64 kByte bei weitem nicht mehr aus. Ein größerer Adreßraum, wie er für 16– und 32–Bit–Mikroprozessoren typisch ist, kann beispielsweise durch Vergrößerung der Adresse auf z.B. 32 Bit bereitgestellt werden, was einem Adreßraum von 4 GByte entspricht.

7.2 Befehlssatz

Die Gesamtheit der von einem Mikroprozessor ausführbaren Befehle wird als Befehlssatz (engl. *instruction set*) bezeichnet. Er umfaßt größenordnungsmäßig 50 ... 100 Grundbefehle, die in der Funktion und auch im Assemblercode für verschiedene Mikroprozessortypen ähnlich sind. Der Befehlssatz, insbesondere bei 16– und 32–Bit–Mikroprozessoren, ist vergleichbar mit dem von Klein- und Minirechnern. Er läßt sich in mehrere Gruppen einteilen. In einigen Gruppen gibt es bei einzelnen Mikroprozessortypen (-familien) nur sehr wenige oder keine Befehle. Mit steigender Wortlänge findet man bei Mikroprozessoren der CISC–Klasse (engl. *complex instruction set computer*, siehe auch Abschnitt RISC–Architektur in Abschnitt 7.5) einen umfangreicheren und mächtigeren Befehlssatz zur Nutzung der erheblich gesteigerten Fähigkeiten der Prozessoren, wobei eine Aufwärtskompatibilität üblich ist: ein 16–Bit–Mikroprozessor enthält den Befehlssatz eines 8–Bit–Mikroprozessors der gleichen „Familie“ als Untermenge; gleiches gilt für 32–Bit–Mikroprozessoren.

In Tabelle 7.2 seien kurz einige Vertreter der bekanntesten Mikroprozessorfamilien mit ihren wichtigsten Daten aufgelistet: die Serien Intel $80x88/80x86$ und Motorola $680xx$.

7.2.1 Datentransportbefehle

Sie bewirken die Übertragung von Daten und Adressen zwischen Registern und Speicher und erfordern die Angabe von Quell- und Zieladresse. Als Lade- und Transportbefehle heißen sie meist MOVE oder LD (load). Zu dieser Gruppe gehören auch Austauschbefehle EX (exchange) für den Austausch des Inhalts zweier Speicherstellen oder Register. Ein Befehl SWAP vertauscht die beiden Bytes eines 16–Bit–Registers miteinander. Mit CLR (clear) kann ein Register gelöscht, d.h. mit Null geladen werden. Wichtig für das Sichern von Registerinhalte beim Sprung zu Subroutinen sind Befehle zum Ablegen von Daten im Stackspeicher (PUSH) und zum Wiederauslesen aus dem Stack (POP) in adressierte Register.

Prozessortyp Intel	Motorola	Register (bit)	Datenbus (bit)	Adreßbus (bit)	Physikal. Adreßraum	Bemerkung
8080		8	8	16	64 kByte	
8088		16	8	20	1 MByte	
8086		16	16	20	1 MByte	
80286		16	16	24	16 MByte	
	68000	32	16	24	16 MByte	
	68020	32	32	32	4 GByte	
80386		32	32	32	4 GByte	MMU[1] integr.
80386SX		32	16	32	4 GByte	MMU integr.
	68030	32	32	32	4 GByte	MMU integr.
80486		32	32	32	4 GByte	MMU, FPU[2] u.
	68040	32	32	32	4 GByte	Cache[3] integr.
Pentium		32	64	32	4 GByte	MMX–Technologie eingef.
Pentium Pro			64	64	64 GByte	L2 Cache integr.
PentiumII			64	64	64 GByte	

Tabelle 7.2: Übersicht über Mikroprozessor–Familien

7.2.2 Blocktransfer- und Suchbefehle

Mit einem Blocktransferbefehl kann der gleichartige Transport von Bytes oder Worten eines Datenblocks oder einer Zeichenkette effektiv programmiert werden. Ein Blocksuchbefehl vergleicht z.B. die Elemente eines Datenblocks, dessen Anfangsadresse in einem Register steht, nacheinander mit dem Akkumulatorinhalt bis zur ersten Übereinstimmung oder bis bei Nichtübereinstimmung das Blockende erreicht ist.

7.2.3 Arithmetikbefehle

Die grundlegenden Arithmetikbefehle sind Addition ADD und Subtraktion SUB von binär codierten Zahlen, und zwar mit einfacher oder doppelter Länge gegenüber der Wortlänge des Mikroprozessors. Bei doppelter Länge (double precision) übernimmt ein Registerpaar die Rolle des Akkumulators. Rechenoperationen mit mehrfacher Wortlänge werden mit Nutzung des Übertragsbits (engl. *carry*) als Unterprogramme realisiert. Zu den Arithmetikbefehlen gehören auch die Befehle für Inkrementieren und Dekrementieren von Register- und Speicherzelleninhalten, gewöhnlich um 1, in Einzelfällen auch um programmierbare Werte im Bereich von 1 bis 16. Der Vergleichsbefehl CMP (compare) entspricht einer Subtraktion, die bei Übereinstimmung den Wert Null ergibt und das Z–Flag setzt (Z = 1). Weitere Befehle beziehen sich auf die Addition und Subtraktion von binär codierten Dezimalzahlen BCD, die als Halbbytes gehandhabt werden.

Bei 16–Bit–Mikroprozessoren sind Befehle für Multiplikation und Division implementiert. Mit zwei 16–Bit–Operanden ist das Produkt 32 Bit lang. Bei einer Division hat der Divident die doppelte Wortlänge, damit als Resultat ein Quotient mit 16–Bit–Genauigkeit geliefert wird. Die Operanden sind ganze Zahlen mit oder ohne Vorzeichen. Eine höhere Rechengenauigkeit ist mit Berücksichtigung des Übertragsbit durch ein entsprechendes Unterprogramm möglich. Dies gilt auch für Gleitkommarechnung (reelle Zahlen), sofern hierfür nicht ein Numerik–Coprozessor zur Verfügung steht.

7.2.4 Logikbefehle

Zum Standardbefehlssatz von Mikroprozessoren gehören Logikbefehle für die bitweise logische Verknüpfung zweier Operanden (Bytes oder Worte) durch UND (AND), ODER (OR) oder Exclusiv–ODER

(XOR). Weiterhin ist die Negation eines Registerinhalts als logischer Befehl implementiert.

7.2.5 Schiebe- und Rotationsbefehle

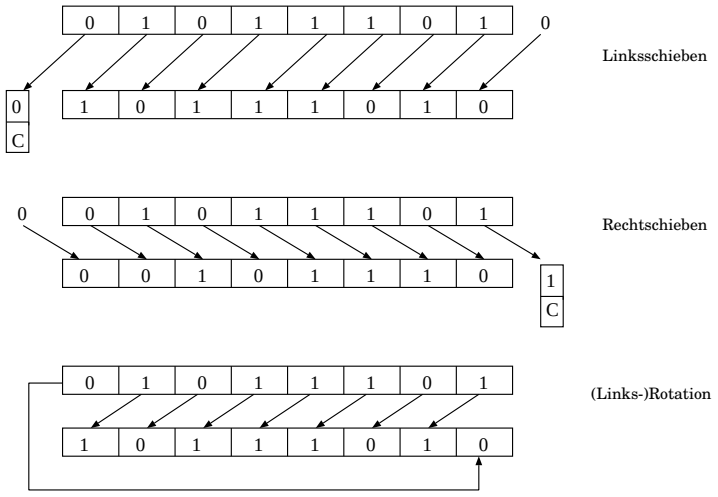


Abbildung 7.4: Darstellung von Schiebe- und Rotationsbefehlen

Sie entsprechen einem Schieberegister oder einem Ringzähler und finden insbesondere bei der binären Multiplikation oder Division Verwendung, Abb. 7.4. Die Verschiebung eines Registerinhalts um eine Position nach links in Richtung höherer Wertigkeit ist gleichbedeutend mit der Multiplikation um den Faktor 2, die Verschiebung nach rechts stellt daher eine Division durch 2 dar. Gleichzeitig muß die freiwerdende niederste bzw. höchstwertige Bitposition mit einer Null gefüllt werden. Bei vorzeichenbehafteten Operanden muß man das Vorzeichenbit entsprechend berücksichtigen (arithmetische Verschiebung). Das Übertragsbit C (engl. *carry*) nimmt das jeweils zuletzt aus dem Register herausgeschobene Bit auf.

Bei der Rotation werden diejenigen Bits nachgezogen, die „am anderen Ende“ herausgeschoben wurden, so daß keine Daten verloren gehen; das Übertragsbit kann dabei einbezogen werden.

7.2.6 Bitverarbeitende Befehle

Sie erlauben das Setzen (SET), Rücksetzen (RES) oder Testen (BIT) einzelner Bitpositionen. Befehle dieser Art, die für viele Aufgaben der Prozessüberwachung vorteilhaft nutzbar sind, sind in den Mikroprozessoren der Intel 80x86–Familie (zu finden in Personal Computern) nicht implementiert.

7.2.7 Ein- und Ausgabefehle

Sie beziehen sich auf den Datentransfer mit peripheren Einheiten über Ein-/Ausgabekanäle (Ports), wobei deren Adreßraum vom Arbeitsspeicher getrennt ist. Die Portadresse kann in den IN– und OUT–Befehlen direkt oder indirekt angegeben sein. Das zu transferierende Byte oder Wort wird einem Register des Mikroprozessors entnommen bzw. bei Eingabe dort abgelegt. Bei den Mikroprozessoren der Motorola 680xx–Familie wird eine speicherbezogene Adressierung durchgeführt, so daß keine expliziten IN– und OUT–Befehle vorhanden sind.

7.3 Steuerung

7.3.1 Programmsteuerung

Diese Befehlsgruppe erlaubt die Programmverzweigung in Abhängigkeit von Bedingungen und ist für die Nutzung von Subroutinen und für die Steuerung von Programmschleifen erforderlich. Bei Sprungoperationen (JMP) wird im Operandenteil das Sprungziel angegeben, wobei der Sprung bedingt oder unbedingt ausgeführt wird. Bei bedingten Sprüngen wie auch bei Ruf- und Rückkehrbefehlen wird die Ausführung vom Zustand ausgewählter Bedingungsbits (Flags) abhängig gemacht.

Beim Aufruf von Subroutinen (CALL) wird unbedingt oder bedingt zu der im Operandenteil enthaltenen Anfangsadresse einer Subroutine gesprungen. Dabei wird die Adresse des im Programm nach CALL genannten Befehls automatisch im Stackspeicher abgelegt.

Nach Abarbeitung einer Subroutine wird zum ursprünglichen Programm zurückgekehrt (RET), indem die Fortsetzungsadresse aus dem Stackspeicher gelesen und in den Befehlszähler geschrieben wird.

7.3.2 Prozessorsteuerung

Hierzu sind schon bei 8–Bit–Mikroprozessoren Befehle zur Steuerung des Befehlsablaufs zu finden. Leer- und Pausenbefehle (NOP — no operation): der Prozessor führt einen Befehls–Zyklus ohne Registerbeeinflussung aus; z.B. für Zeitverzögerungen. Haltbefehl HALT: der Prozessor stoppt die weitere Programmausführung und wartet auf ein Interruptsignal, um das Programm fortzufahren; z.B. zur Synchronisation des Programmablaufs mit externen Ereignissen. Interrupterlaubnis: im einfachsten Fall wird der maskierbare Interrupteingang des Prozessors mit dem Befehl EI (enable interrupt) geöffnet und mit DI (disable interupt) blockiert.

Weitere, von der Struktur des Prozessors abhängende Steuerbefehle beeinflussen den Interruptmodus, erlauben die Veränderung von Statusregistern, realisieren den Aufruf des Systemmodus usw.

7.3.3 Systemsteuerung

Hierunter werden Befehle verstanden, die für das Zusammenwirken des Prozessors mit Erweiterungseinheiten (z.B. Numerik–Coprozessoren, Speicherverwaltungseinheiten, Cache–Controler) und für die Bussteuerung bei Multiprozessorsystemen von Bedeutung sind. Sie sind bei 16– und 32–Bit–Mikroprozessoren implementiert und sind im Zusammenhang mit der Prozessor- und Rechnerarchitektur zu sehen. Einige Details hierzu findet man im Abschnitt 7.5.

7.4 Der 8–Bit–Mikroprozessor Intel 8080

Der 8080 ist ein Prozessor mit einer Wortbreite von 8 Bit. Er wurde von der Firma Intel erstmals im Jahre 1973/74 produziert und enthält 4800 Transistoren auf einer Fläche von etwa 22 mm^2 . Die interne Struktur des 8080 ist schematisch in Bild 7.1 dargestellt. Er hat 16 Adreßleitungen für einen Adreßraum von 64 kByte und eine Befehlszykluszeit von etwa $2\mu s$. Die Belegung der 40 Anschlußkontakte und die Registerstruktur sind in Abb. 7.5dargestellt.

Die Befehlsausführung erfolgt im Prinzip gemäß Bild 7.3 mit 1 bis 5 Maschinenzyklen M, die jeweils aus 3 bis 5 Maschinentakten T sowie gegebenfalls zusätzlichen Wartetakten T_w bestehen.

Der Datentransfer auf dem bidirektionalen Datenbus D7 ... D0 wird von Steuersignalen begleitet. Während des Synchronisiersignals SYNC, das zu Beginn jedes Maschinenzyklus aktiviert wird (SYNC = 1),

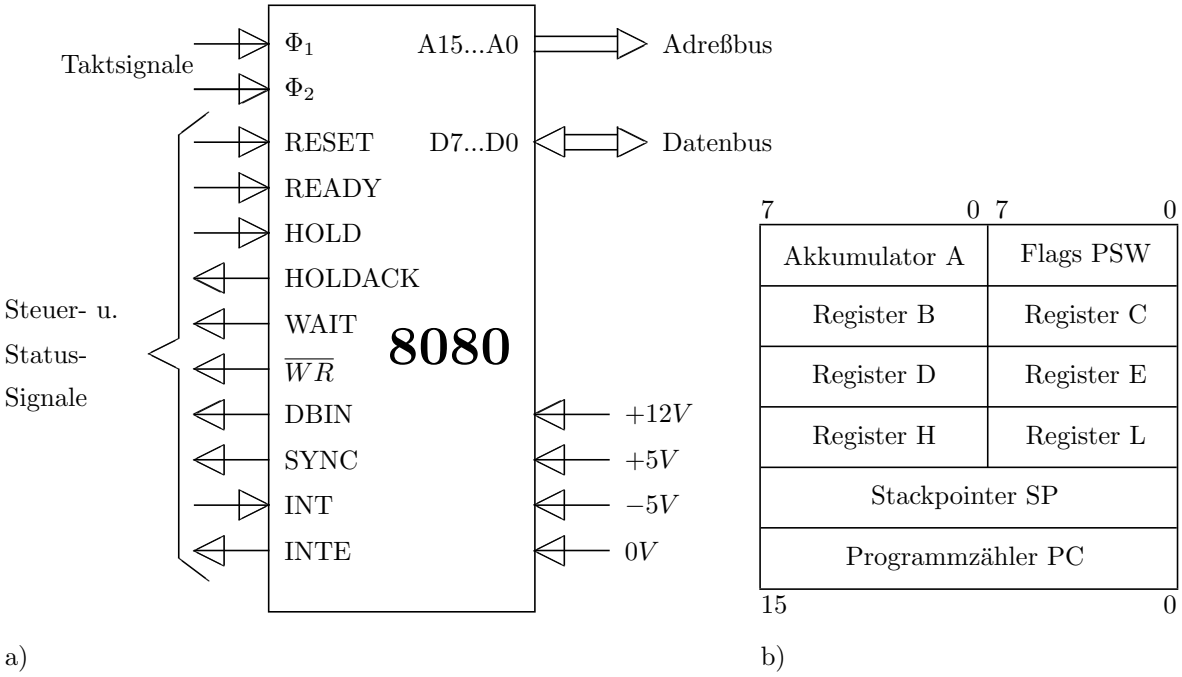


Abbildung 7.5: Mikroprozessor 8080; a: Anschlüsse; b: Registerstruktur

führt der Datenbus eine Statusinformation über den momentan laufenden Maschinenzyklus, die von den angeschlossenen Logikschaltkreisen zur Steuerung der Speicher– und Ein-/Ausgabe–Register genutzt werden kann. Die Signale DBIN (data bus in) und $\overline{\text{WR}}$ (write) zeigen bei DBIN = 1 Dateneingabebereitschaft und bei $\overline{\text{WR}} = 0$ stabile Ausgabedaten auf dem Datenbus an. Im Maschinenzyklus T_2 wird das Eingangssignal READY getestet. Bei READY = 0 werden Wartezustände T_w bis zum Vorliegen gültiger Daten aus dem Speicher oder von einem externen Register eingefügt. Den Wartezustand zeigt der Prozessor mit WAIT = 1 an.

Eine Interruptanforderung am maskierbaren Eingang INT wird nach Ablauf der Befehlsausführung mit dem Statussignal INTA zeitsynchron zu SYNC quittiert. Hiermit kann die Eingabe eines Restart–Befehls (RST) gesteuert werden, der den Beginn der Interruptbedienung je nach Kodierung bei einer von 8 Adressen 0, 8, 16, ..., 56 (38_{16}) veranlaßt. Voraussetzung für die Annahme einer Interruptforderung ist allerdings, daß sich das Interrupt–Enable–Flipflop im Zustand „Enable“ befindet. Dieser Zustand, der am Prozessoranschluß INTE getestet werden kann, wird mit dem Befehl EI (enable interrupt) erreicht.

Ein Eingangssignal HOLD fordert einen Betriebszustand des Mikroprozessors an, bei dem er die Daten- und Adreßbusanschlüsse in den hochohmigen Zustand schaltet. Damit ist der Mikroprozessor vom Bus elektrisch abgetrennt, um z.B. einer Steuerung den Bus für einen direkten Speicherzugriff überlassen zu können. Der HOLD–Zustand wird mit HLDA angezeigt.

Der RESET–Eingang, der nach Einschalten der Betriebsspannungen oder zur Wiederherstellung eines definierten Programmzustands aktiviert wird, schaltet den Befehlszähler auf Null (Programmstart nach RESET ab Speicherzelle 0) und blockiert den Interrupteingang, beeinflußt aber nicht die Registerinhalte des Mikroprozessors. Der Mikroprozessor 8080 benötigt einen externen 2–Phasen–Takt und die Betriebsspannungen + 5 V, - 5 V und + 12 V. Diese Anforderungen stellen einen Mangel dar, der im Interesse eines einfacheren Mikrorechneraufbaus bei anderen Mikroprozessortypen überwunden wurde.

Die 10 Anschlüsse für die Steuerfunktionen reichen nicht aus. Daher müssen einige Steuersignale über den Datenbus im Zeitmultiplexbetrieb ausgegeben werden. Diese acht zusätzlichen Statusfunktionen (z.B. INTA) erscheinen zu bestimmten Zeitpunkten auf dem Datenbus.

Der für den Programmierer in Assemblersprache wichtige Registerblock (Abb. 7.5b) enthält sechs 8–Bit–Register für freiprogrammierbare Nutzung (Register B, C, D, E, H, L), den Akkumulator (A) sowie den Programmzähler (PC) und den Stack–Pointer (SP), beide mit 16 Bit Länge. Der Stack kann an jeder beliebigen Stelle im Hauptspeicher angelegt werden. Die Flexibilität des Stackmechanismus beim 8080 erlaubt eine nahezu beliebige Schachtelung von Unterprogrammaufrufen. Die Register lassen sich für einige Operationen mit 16 Bit Wortlänge als Registerpaare BC, DE und HL programmieren. Die indirekte Registeradressierung nutzt das Registerpaar HL zum Ablegen der 16 Bit umfassenden Speicheradresse. Das Flag- oder Zustandsregister (PSW — Prozessor–Status–Wort) gibt den Prozessorzustand oder das Ergebnis nach Ausführung von Befehlen an:

- Null (Z, zero): gesetzt (Z = 1), wenn das Resultat des ausgeführten Befehls Null ergeben hat,
- Vorzeichen (S, sign): gesetzt bei Vorzeichenbit = 1 (minus),
- Parität (P, parity): gesetzt, wenn die Summe der mit 1 belegten Bitpositionen eine gerade Zahl ergibt (parity even),
- Übertrag (C oder CY, carry): gesetzt, wenn nach einer arithmetischen Operation ein Übertrag entstanden ist.

Ein weiteres Zustandsbit ist der Hilfsübertrag (AC, auxiliary carry), der einem Übertrag vom Bit 3 zum Bit 4 entspricht und vor allem bei BCD–Operationen angewendet wird, aber im Gegensatz zu den anderen Zustandsbits nicht für bedingte Sprung- und Rufoperationen nutzbar ist.

7.5 Architekturerweiterung und moderne Konzepte

Seit der Einführung der 8–Bit–Mikroprozessoren sind aufgrund erheblich gestiegener Anforderungen und begünstigt durch große Fortschritte in der Herstellungstechnik die Leistungsdaten der nachfolgend eingeführten Mikroprozessoren drastisch verbessert worden. Dies wurde erreicht durch Erweiterung bestehender Mikroprozessor–Architekturen (z.B. größere Wortbreite), durch Übernahme von Techniken aus dem Mini- und Großrechnerbereich (z.B. virtuelle Speichertechnik), sowie durch Einführung neuer Konzepte (z.B. RISC — engl. **r***educed* **i***nstruction* **s***et* **c***omputer*). Darüberhinaus werden in Zukunft verstärkt neue Materialien auch im Mikrorechnerbereich eine Rolle spielen (z.B. Gallium–Arsenid — GaAs).

Die wesentlichen Verbesserungen durch Erweiterungen betreffen folgende Merkmale: (*kursiv* gedrucktes wird im Anschluß näher erläutert)

- *Vergrößerung der Wortbreite* auf 16 Bit und 32 Bit und 64 Bit.
- *Vergrößerung des Adreßraums* auf 1 MByte (2^{20}), 16 MByte (2^{24}) oder 4 GByte (2^{32}) und erweiterte Adressierungsarten.
- umfangreicherer und allgemein verwendbarer Registersatz, größere Breite für Daten und Adreßberechnung.
- erweiterte Befehlsliste mit weitgehender Aufwärtskompatibilität zu den 8–Bit–Mikroprozessoren, u.a. Befehle für Multiplikation/Division, Systemfunktionen (privilegierte Befehle), Steuerfunktionen.
- Maßnahmen zur Datensicherung und zum Schutz von Systemfunktionen gegen Fehlbedienung, z.B. durch Unterscheidung zwischen System- und Normal–Betriebsart des Mikroprozessors.
- vielseitiges Interrupt- und Trapsystem zur Behandlung externer Interruptanforderungen und bestimmter Zustände des Programmablaufs.

- höhere Taktraten.

Zu den aus dem Mini- und Großrechnerbereich eingeführten Techniken zählen insbesondere

- spezielle *Adressierungstechniken* (Segmentierung, Seitenadressierung, virtuelle Speichertechnik) mit Befehlen zu ihrer Unterstützung.
- Realisierung von *Speicherverwaltungseinheiten* (MMU).
- Multitaskingfähigkeit mit Befehlen zur Unterstützung der Schutzfunktionen des Betriebssystems (z.B. gegenseitige Sicherung der Speicherbereiche verschiedener Tasks).
- Interne *Warteschlangen* für Befehle, Daten und Adresse.
- Interne oder externe Befehls– und Daten–*Caches* (schnelle Zwischenspeicher) zwischen CPU und Arbeitsspeicher für schnellen Zugriff (auch: zwischen Arbeits- und externen Speicher, z.B. Platte).
- *Parallel- oder Fließbandverarbeitung* von mehreren Befehlen durch gleichzeitige Arbeit mehrerer Funktionseinheiten des Mikroprozessors, auch unter Einschluß von Coprozessoren.
- *Multiprozessorsysteme* mit Parallelarbeit von mehreren gleichberechtigten Prozessoren (z.B. Transputer).

Im folgenden soll auf einige wichtige Merkmale näher eingegangen werden.

7.5.1 Erhöhung der Wortbreite

Die Erhöhung der Wortbreite auf z.B. 32 oder 64 Bit bringt den Vorteil eines höheren Datendurchsatzes und für arithmetische Operationen eine höhere Genauigkeit bei meist gleichbleibender Ausführungszeit. Voraussetzung ist, daß dabei 4 bzw. 8 Bytes aus aufeinanderfolgenden Speicherstellen als 1 Wort übertragen werden können, was eine Datenbusbreite von 32 bzw. 64 Bit zwischen CPU und Arbeitsspeicher erfordert.

Ist die Datenbusbreite geringer als die Wortbreite des Prozessors (z.B. 8 Bit Datenbus beim 16–Bit–Mikroprozessor Intel 8088), erfordert der Transport von Daten mehrere Maschinenzyklen (hier: 2) und verlängert folglich die Befehlsausführungszeit.

7.5.2 Vergrößerung des Adreßraums

Durch Vergrößerung des Adreßraums lassen sich erheblich größere Arbeitsspeicher für umfangreiche System- und Anwendungsprogramme adressieren und externe Datenspeicher besser nutzen. Diese und weitergehende Forderungen nach verschieblichen (relocatable) Programmen und nach Schutzfunktionen für Betriebssystem und Anwenderprogramme führten zur Realisierung einer Speicherverwaltung (memory management). Ein größerer Adreßraum für Programme erfordert zunächst eine längere Adresse, z.B. 20 Bit für 1 MByte Adreßraum oder 24 Bit für 16 MByte. Im einfachsten Fall entspricht jeder logischen Adresse eine gleichlautende physikalische Adresse des Arbeitsspeichers. Diese bei 8–Bit–Mikroprozessoren mit 16–Bit–Adresse übliche Konfiguration ist bei beträchtlich größeren Adreßräumen aus Kostengründen nicht mehr realisierbar und auch nicht unbedingt die zweckmäßigste Lösung. In derartigen Rechnern ist dann eine **Speicherverwaltungseinheit (MMU)** notwendig. Sie ist zwischen CPU und Speicher angeordnet und transformiert den höherwertigen Adreßteil, womit das Programm im physikalischen Adreßraum verschoben wird. Mit der Speicherverwaltung können also logische und physikalische Adreßräume unterschiedlicher Größe aufeinander abgebildet werden.

Die Abbildung eines kleineren logischen Adreßraumes auf einen größeren physikalischen Adreßraum ist im einfachsten Fall durch eine **Speicherbankumschaltung** möglich, bei der z.B. mit einem logischen Adreßbereich von 64 kByte mehrere 64 kByte umfassende Speicherblöcke verfügbar sind.

Erwähnt werden soll hier auch noch die **Overlay–Technik**, bei der der Arbeitsspeicher durch einen externen Speicher ergänzt wird. Dabei wird ein Programm vom Programmierer in Gruppen von solchen Unterprogrammen zerlegt, die jeweils gleichzeitig im Hauptspeicher sein müssen. Das Hauptprogramm bleibt ständig im Arbeitsspeicher. Sprünge auf ein zur Zeit nicht speicherresidentes Unterprogramm werden vom Betriebssystem abgefangen und laden das benötigte Programm, wobei nicht mehr benötigte Programme zunächst aus-, dann überlagert werden. Der Rechneranwender braucht sich dabei nicht um die Transformationstabellen zwischen logischer und physikalischer Adresse zu kümmern.

Seitenadressierung und **Segmentierung** unterscheiden zwei Teile der logischen Adresse. Der obere Adreßteil beschreibt die Seiten– bzw. Segment–Adresse und der untere Adreßteil gibt die Wortadresse innerhalb der Seite bzw. des Segments an. Während bei Seitenadressierung die Seiten konstante Längen haben, kann die Segmentlänge der Programmsegmentlänge angepaßt werden.

Die Aufteilung der Adresse vergrößert den insgesamt verfügbaren Adreßraum nicht, bewirkt aber eine Verkürzung der Adreßlänge im Befehl. Erst die Zwischenschaltung einer Adreßtabelle bewirkt eine Verlängerung der Adresse und damit eine Vergrößerung des Adreßraums.

Bei beiden Methoden kann durch Austausch der Adreßtabellen dem Programm ein anderer Satz von Seiten oder Segmenten zugänglich gemacht werden. Damit wird der Multitask–Betrieb des Rechners unterstützt, und jede Task kann den vollen logischen Adreßraum ausnutzen. Selbstverständlich müssen gewisse Schutzfunktionen (z.B. Zugriffsrechte) die unerwünschte Beeinflussung zwischen unterschiedlichen Tasks ausschließen.

7.5.3 Virtuelle Speichertechnik

Eine elegante und auch im Mikrorechnerbereich an Bedeutung gewinnende Technik ist die virtuelle Speichertechnik. Sie ermöglicht die Nutzung eines sehr großen virtuellen Speicherraums (z.B. 4 GByte bei 32–Bit–Prozessoren), ohne daß sich der Programmierer um die physikalisch realisierten Adressen kümmern muß. Greift das Benutzerprogramm auf eine Adresse zu, wird diese von der MMU auf eine reale Speicher–Adresse umgerechnet. Befindet sich die angeforderte Seite oder das Segment nicht im Arbeitsspeicher, erzeugt die Speicherverwaltung ein Fehlersignal, welches die weitere Befehlsabarbeitung abbricht. Die dabei ausgelösten Aktionen werden vom Prozessor und vom Betriebssystem ausgeführt: Retten des aktuellen Prozessorzustandes, Einlesen des Speicherbereichs in den Arbeitsspeicher, gegebenenfalls noch Auslagern nicht mehr benötigter Programme und Daten auf den externen Speicher (swapping) und Fortsetzen des Programms. Voraussetzung ist die Möglichkeit des definierten Befehlsabbruchs (abort), um nach Bereitstellung der Seite oder des Segments diesen Befehl wiederholen oder beenden zu können.

7.5.4 Zeitlich verschränkte Befehlsausführung

Prinzipiell umfaßt die Befehlsausführung unterschiedliche Phasen: Befehl–Lesen, Adressberechnung, Operanden–Lesen, Ausführung. Für einen Einzelbefehl werden diese Phasen zwar nacheinander durchlaufen, doch können gleichzeitig mehrere aufeinanderfolgende Befehle in unterschiedlichen Phasen in Arbeit sein. Man nennt diese Methode **Fließband- oder Pipelineverarbeitung.** Voraussetzung dafür ist, daß mehrere Funktionseinheiten des Mikroprozessors gleichzeitig arbeiten können. Pipelining wird eingesetzt in Verbindung mit internen Warteschlangen für Befehle, Adressen und Daten, die durch prozessorinterne oder extern realisierte Befehls– und Daten–Caches ergänzt werden können.

So arbeiten zum Beispiel im Fall des Intel 8086 das Bus–Interface BIU und die Ausführungseinheit EU zeitlich parallel. Das BIU enthält für die vom Arbeitsspeicher eingelesenen Bytes ein 6 Byte umfassendes **Warteschlangenregister**, welches als FIFO–Speicher (first–in–first–out) 1 Wort (2 Byte) an die

EU abgibt, sobald diese zur Übernahme bereit ist. Während der Abarbeitung in der Ausführungszeit liest das Businterface ein weiteres Wort in das Warteschlangenregister ein, so daß stets mehrere Worte zur Ausführung bereit stehen. Die Fließbandverarbeitung steigert die Busauslastung und reduziert den Einfluß der Speicherzugriffszeit auf den Datendurchsatz, da die Ausführungseinheit nicht mehr auf die Speicherdaten warten muß. Für die Länge der Warteschlange gibt es ein prozessorabhängiges Optimum, das beim 8086 bei 6 Byte liegt. Da bei allen Sprung- und Verzweigungsbefehlen die Warteschlange gelöscht und neu gefüllt werden muß, würde eine zu lange Warteschlange zu einer erheblichen Anzahl nutzloser Busoperationen führen.

7.5.5 Cache

In ähnlicher Weise wie Warteschlangen arbeiten interne oder externe Caches. Sie beruhen auf der Überlegung, daß die nachfolgende Befehls- oder Datenadresse meist in der Nähe der momentanen aktuellen Speicheradresse liegt. Da die Zugriffszeit des Arbeitsspeichers mit einer Größenordnung von 100 ns die Verarbeitungsgeschwindigkeit der zum Teil wesentlich schnelleren CPU negativ beeinflußt, ordnet man zwischen Prozessor und Arbeitsspeicher einen schnellen Zwischenspeicher aus statischem RAM (im Bereich wenige kByte–1 MByte) mit Zugriffszeiten bei 20–40 ns an. Mit speziellen Busoperationen (burst mode) ohne Wartezyklen wird bei einem Zugriff auf den Arbeitsspeicher gleichzeitig ein ganzer Datenblock von z.B. 16 aufeinanderfolgenden Bytes in den Cache geladen. Damit besteht eine gewisse Chance, daß der oder die nachfolgenden Befehle und/oder Daten direkt aus dem Cache in den Prozessor übertragen werden können. Im günstigen Falle stehen ganze Programmschleifen und die zugehörigen Daten im Cache.

Caches werden oft auch zwischen Arbeits- und externem Speicher (Festplatte) verwendet. Die Größe des Plattencaches kann einige 100 KByte betragen und beschleunigt die Plattenzugriffe zum Teil erheblich. Auch hier gelten die oben gemachten Überlegungen, daß Programmcode und Daten meist in der Reihenfolge benötigt werden, wie sie abgespeichert sind. Daher werden bei einem Plattenzugriff gleichzeitig mehrere aufeinanderfolgende Sektoren oder ganze Spuren in den Cache geladen. Durch Optimierung der Cacheverwaltung lassen sich Trefferquoten (cache hit rate) von zum Teil weit über 90% erzielen, wobei die Größe des Caches eine wichtige Rolle spielt. Im Bereich der Personal Computer wird oft bereits auf dem Plattencontroler ein Cache hardwaremäßig integriert.

7.5.6 Parallelverarbeitung

Der Aspekt der Parallelverarbeitung ist schon bei der Zusammenarbeit von Prozessor und Numerik–Coprozessor angeklungen. Hier übernimmt der Prozessor z.B. die Befehlsdecodierung und Adreßberechnung sowie die Ausführung von Integer–Arithmetik, während der Coprozessor die komplizierte Fließkomma–Arithmetik übernimmt. Die beiden Prozessoren sind jedoch nicht gleichberechtigt. Eine strengere Definition von Parallelverarbeitung spricht von gleichzeitiger Verarbeitung eines Programms durch mehrere (gleichberechtigte) Prozessoren. Bei n Prozessoren ergibt sich in diesem Fall eine Beschleunigung der Verarbeitung um den Faktor n. Dies setzt jedoch voraus, daß sich das Programm so aufteilen läßt, daß die einzelnen Prozessoren ihre Teilaufgaben (Tasks) unabhängig voneinander bearbeiten können. Andernfalls müssen Prozessoren auf die Ergebnisse anderer Prozessoren warten. Methoden für die Zerlegung von Programmen existieren bisher kaum. Auch parallele Programmiersprachen befinden sich noch weitgehend in der Entwicklung.

Während der Abarbeitung müssen die einzelnen Tasks miteinander kommunizieren können, damit gegenseitige Synchronisation erreicht wird, Parameterwerte ausgetauscht werden können und Zugriff zu globalen Systemressourcen möglich ist. Neben der erheblichen Komplizierung der Software in Entwurf und Struktur müssen Betriebssysteme und Hardware die Parallelisierung unterstützen. Ein Mikroprozessor, der speziell für Parallelverarbeitung entwickelt wurde, ist der **Transputer**. Die Zielsetzungen waren hohe Verarbeitungsleistung und Geschwindigkeit für den Einsatz in Multiprozessoranwendungen

und hohe Flexibilität, gekoppelt mit problemloser Einsatzmöglichkeit. Ein weiterer wichtiger Aspekt war, die Unterstützung einer höheren Programmiersprache durch den Prozessor zu erreichen. Hohe Verarbeitungsleistung wurde erreicht durch Übernahme von Konzepten der RISC–Architektur (siehe unten), und um das Problem der Zugriffszeiten auf externen Speicher zu umgehen, wurde ein schnelles RAM auf dem Prozessor integriert. Die Kommunikation zwischen Transputern in Multiprozessorsystemen wurde durch sogenannte Links erreicht, schnelle serielle Schnittstellen mit eigenen DMA–Controllern. Um ein hohes Maß an Parallelisierung zu erreichen, muß die Kommunikation zwischen den einzelnen Prozessoren optimiert werden. Das vorher verwendete Konzept, Prozessoren über einen gemeinsamen Bus zu verbinden, scheitert am Kommunikations–Overhead, der durch die notwendige Verständigung der Prozessoren über den Bus entsteht. Ab einer gewissen Anzahl von Prozessoren (ca. 4–6) läßt sich die Verarbeitungsgeschwindigkeit nicht mehr steigern; sie sinkt im Gegenteil dann, wenn die Prozessoren nur noch mit Verwaltungsoperationen beschäftigt sind. Der Fortschritt beim Transputer war die autonome Arbeit der Kommunikation auf dem Chip bei gleichzeitigem Datenaustausch. Das ermöglicht die Kopplung einer fast beliebigen Anzahl, so daß man bei linearer Leistungszunahme Systeme von mehreren tausend Transputern aufbauen kann. Das Problem der Programmierung von Transputern in solch hochkomplexen Rechnerstrukturen wurde durch Konzeption einer Hochsprache gelöst, die den Anforderungen der Parallelverarbeitung gerecht werden soll: OCCAM.

Der Prototyp des ersten Transputers T414 der Firma Inmos wurde 1984 entwickelt, der Chip wurde 1986 ausgeliefert. Er besitzt 4 Links, über die weitere Transputer angeschlossen werden können. Der 32–Bit–Prozessor wird getaktet mit 5 MHz, die intern auf 15 oder 20 MHz erhöht werden. Über den internen Bus kann er direkt ohne Wartezyklen auf das statische 2–kByte on–chip–RAM zugreifen. Der Transputer besitzt einen mikrocodierten Prozessorverteilungsmechanismus (process scheduler), der es einer beliebigen Anzahl von Prozessen erlaubt, gleichzeitig (quasi–parallel) auf dem Transputer abzulaufen und dessen Leistung unter sich aufzuteilen. Das erspart einen weitaus zeitaufwendigeren Software–Verteilungsmechanismus. Der 1987 vorgestellte Nachfolger des T414, der T800 ist anschluß- und softwarekompatibel. Er unterscheidet sich vom Vorgänger im wesentlichen in 3 Punkten: integrierte Fließkomma–Recheneinheit, kompletter Memory–Controler und auf 4 kByte erhöhte RAM–Kapazität. Außerdem wurde die Taktrate erhöht. Transputer können in fremde Rechnersysteme integriert werden. So gibt es Erweiterungskarten für PC's, mit denen man den PC–Prozessor durch sehr leistungsfähige Multi–Transputersysteme ergänzen kann. Wegen der unterschiedlichen Maschinensprachen muß der Programmcode jedoch prozessorspezifisch compiliert werden.

7.5.7 RISC–Architektur

Die stetige Weiterentwicklung der Mikroprozessoren, vor allem im Hinblick auf Kompatibilität zu ihren Vorgängertypen wurde geleitet von folgenden Zielen:

- Erweiterung des Funktionsspektrums von höheren Programmiersprachen durch Schaffung von flexiblen Compilerinstruktionen, die direkt in die Hardware implementiert werden konnten.
- Die Verschiebung der zeitintensiven Teile des Betriebssystems in den Mikrocode.
- Die Zurverfügungstellung anwendungsorientierter Befehle, die der Anwender selbst in Mikroprogramme implementieren konnte.

Die so entstandenen Computer zeichneten sich durch ein breites Spektrum von relativ komplexen Instruktionen, bestehend jeweils aus mehr als 200 Instruktionen auf der Assemblerebene aus. Diese Architektur wird mit CISC (complex instruction set computer) bezeichnet. Sie hat unter anderem folgende Vorteile:

- Mikrocode–Implementierungen sind schneller als entsprechende Softwarebefehle in Programmen.

- Mikroprogramme gestatten die Implementierung von komplexen Instruktionen, die sowohl den Anforderungen an höhere Programmiersprachen, als auch an Anwendungssoftware (u.a. Verkürzung der Programme) Rechnung trugen.

Untersuchungen den US–Universitäten Berkeley und Stanford Ende der 70er Jahre zeigten jedoch die Leistungsgrenzen der CISC–Architekturen:

- Viele komplexe Instruktionen werden in der Praxis nur selten angewendet.
- Komplexe Instruktionen erfordern relativ aufwendige Codierung und Decodierung.
- Kostenintensive Realisierung der CISC-Architektur, da im Prozessorchip viele Funktionen und Unterstützungsmittel implementiert werden müssen.

Der entscheidende Nachteil ist, daß CISC–Prozessoren in rund 80 Prozent aller Anwendungen lediglich 20 Prozent der vorhandenen Befehle nutzen, wobei die einfachsten Befehle am häufigsten verwendet werden. Dies führte schließlich zu den RISC– (reduced instruction set computer) Architekturen, deren wesentliche Merkmale sind:

- Verwendung einer kleinen Zahl von einfachen Befehlen (bis ca. 100), die die Herstellung einfacher Chips gestattet.
- Befehle haben ein einfaches und festes Format, was einfache und schnelle Decodierung des Operationscodes ermöglicht. Dies gestattet den Verzicht auf Mikroprogramme durch Befehlsimplementierung in Hardware.
- Abarbeitung der Befehle in einem einzigen Maschinenzyklus
- Beschränkung der Kommunikation des Prozessors mit dem Speicher auf Laden und Speichern. Alle anderen Manipulationen geschehen innerhalb der Register des Prozessors. Das vereinfacht die Adressierung und erfordert eine große Zahl von Registern (bis zu 1000) auf dem Chip.
- RISC–Maschinen erfordern einen größeren Compilationsaufwand als CISC–Maschinen. Aufgrund der geringen Anzahl von Befehlen und Adressierungsarten muß mehr Aufwand in Compiler gesteckt werden, die die einfachen Befehle möglichst optimal ordnen und pipelining ermöglichen.

Ein Beispiel für einen RISC–Prozessor ist der ebenfalls für Multiprozessorsysteme geeignete Intel i860, der Anfang 1989 vorgestellt wurde. Er besitzt als erster Mikroprozessor eine Rechenleistung, die mit den ersten Supercomputern der Klasse CRAY vergleichbar ist. Der i860 kann in einem Taktzyklus bis zu 3 Befehle ausführen. Das könnten z.B. je eine Gleitkomma–Addition und eine –Multiplikation sowie ein Integerbefehl sein. Damit erreicht der Prozessor bei einem Takt von 40 MHz im günstigsten Fall 80 MFLOPS und gleichzeitig 40 Mio. Integerbefehle, was eine Gesamtleistung von 120 MIPS (million instruction per second) ergibt. Zum Vergleich: Ein Transputer erreicht bei 20 MHz etwa 10 MIPS und 2,2 MFLOPS. Die große Leistung wird beim i860 durch den 64–Bit–Datenbus und einer internen Pipeline–Architektur erreicht, wie sie von einer CRAY bekannt ist. Der aus 9 Funktionseinheiten bestehende Prozessor i860 vereinigt auf einem Chip die Fähigkeiten eines Supercomputers mit denen einer 3D–Grafik–Workstation, da auch eine Grafikeinheit implementiert ist. Sie unterstützt mit Hilfe einer eigenen Integer–Logik dreidimensionale Darstellung von Körpern und deren Manipulation im Bildspeicher. Neben der Kerneinheit, die den gesamten internen Ablauf steuert und u.a. auch die Integerbefehle verarbeitet, gibt es die FPU, die aus Fließkomma–Controler, –Addierer und –Multiplizierer besteht und 64–Bit–Fließkommazahlen verarbeitet. Eine Speicherverwaltungseinheit, ergänzt um einen 4–kByte großen Befehlscache und einen 8 kByte großen Datencache sowie eine Buskontroll–Einheit sorgen für die Anbindung des Prozessors an den Arbeitsspeicher. Es sind PC–Erweiterungskarten und Compiler im Handel, die aus einem gewöhnlichen PC eine „Cray auf dem Schreibtisch“ machen. Die Leistungsmerkmale solcher RISC–Prozessoren haben mehr und mehr den Markt für Personal Computer verändert.

7.5.8 Materialien

Abschließend soll noch auf die Verwendung anderer **Materialien** hingewiesen werden. **Optische Schaltkreise**, die allerdings erst ganz am Anfang ihrer Entwicklung stehen, können mit nahezu Lichtgeschwindigkeit schalten und versprechen somit enorme Leistung bei geringen Verlusten. Optische Computer werden jedoch in diesem Jahrzehnt keine Rolle spielen.

Aus dem reinen Forschungsstadium heraus und schon konkret in der Anwendung sind integrierte Schaltkreise auf der Basis von **Gallium–Arsenid (GaAs)**; Standard–Halbleitermaterial ist bisher Silizium. GaAs ist ein Halbleitermaterial, das Betrieb bei höheren Frequenzen, geringere Leistungsaufnahme, größere Strahlungsfestigkeit und bessere Temperaturwiderstandsfähigkeit als Silizium bietet. Auf der Jagd nach größerer Rechengeschwindigkeit ist GaAs ein vielversprechendes Grundmaterial, das 5 bis 6 mal schneller als Sizilium ist, ohne radikale Änderungen in Computerarchitektur oder Entwurf zu erfordern. GaAs mag noch nichts für Mikrocomputer sein, aber es wird sowohl in Supercomputern als auch zur Überwindung von Architektur–„Flaschenhälsen“ in allen Rechnern, vom Supercomputer bis zur Workstation, eingesetzt. Dem breiten Einsatz von GaAs stehen bisher höhere Kosten (Faktor 100 zu Silizium) sowie Produktionsschwierigkeiten sowohl bei den Kristallen als auch beim Aufbau der Schaltkreise (durch Dotieren und Aufbringen von Leiterbahnen) entgegen. Überdies ist die ganze Herstellungstechnologie auf Silizium ausgerichtet. Somit hat GaAs viele grundlegende Vorteile gegenüber Silizium, es hat aber auch viele praktische Nachteile. Beispiele für die Anwendung sind der zunächst für das Jahr 1990 angekündigte Höchstleistungsrechner Cray–3, dessen Prozessormodule vollständig aus GaAs sein sollen. Für die Verbindung zwischen Prozessor und Speicher in kleineren Systemen gibt es Logigschaltkreise mit 4 kByte statischem RAM die eine Schreib-/Lesezykluszeit von 3,5 ns haben und 4 kByte ROMs, die in Systemen mit 1 GHz Taktfrequenz funktionieren. „In wenigen Jahren wird die GaAs–Technologie weiter verbreitet sein und wird den Unterschied zwischen Workstations, Groß- und Superrechnern verwischen — die Leistung wird in ähnlichen Größenordnungen sein, und der Unterschied wird dann in der Peripherie und im Speicher zu finden sein“ ([1, 01/90]: The high–octane semiconductor). [4]

7.6 Zusammenfassung

Ein Mikroprozessor (MP) ist die auf einem Integrierten Schaltkreis untergebrachte CPU eines Mikrorechners. Der MP arbeitet alle Befehle ab, er ist in verschiedene Funktionsblöcke unterteilt. Das Datenwerk besteht aus der Arithmetisch–Logischen Einheit (ALU), in ihm werden Daten bearbeitet. Das Adreßwerk besteht aus dem Befehlszähler und den Areßregistern, es dient der Adreßberechnung. Das Steuerwerk ist mit allen Bausteinen über Steuerleitungen verbunden. Es besteht aus dem Befehlsdekodierer und den Befehlsregistern und dient der Befehlsausführung. Der Bustreiber dient der Behandlung des Prozessorbusses. Register sind sehr schnelle prozessorinterne Zwischenspeicher.

Für die Adressierung des Speicherbereiches des Prozessors und seine Verwaltung existieren eine Reihe von Adressierungs- und Verwaltungsarten. Zur Nutzung des sehr großen logischen Adreßraums moderner Prozessoren wird die virtuelle Speichertechnik verwendet. Dabei sorgt eine Speicherverwaltungseinheit für die Bereitstellung des entsprechenden Adreßbereiches im physikalisch adressierten Arbeitsspeicher.

Bei der Abarbeitung eines Befehls werden seine einzelnen Bytes nacheinander in den Dekodierer geholt. Das Steuerwerk sorgt dann per Mikroprogramm für den weiteren Ablauf des Befehls. Der Befehlssatz eines Prozessors umfaßt unter anderem arithmetische Operationen für die verschiedenen Datentypen, logische Operationen, Operationen zur Datenübertragung, Sprungoperationen, sowie Befehle zur Prozessor- und Systemsteuerung. Zu den Srungoperationen gehören neben den unbedingten, bedingten und Unterprogramm–Sprüngen auch die Interrupts. Letztere sorgen vor für den Rücksprung und die Wiederaufnahme des unterbrochenen Prozesses, indem sie wesentliche Informationen auf dem Stack ablegen.

[4] W.W. Baumann, Januar 1991

Als Beispiel für einen MP haben wir den Intel 8080 kennengelernt. Er ist ein 8–Bit Prozessor mit 64 kByte Adreßraum und der Urvater aller Prozessoren in den MS–DOS–Rechnern.

Die ständige verbesserte Leistung neuerer MP wird erreicht durch Erweiterung der bestehenden Architektur (größere Wortbreite, größerer Adreßraum), durch Übernahme von Techniken aus dem Großrechnerbereich (virtuelle Speichertechnik, Pipelining, auch in Verbindung mit Warteschlangen und Caches), sowie durch Einführung neuer Konzepte (RISC, Transputer).

7.7 Fragen und Übungen zu Kapitel 7

F 7.1 Was ist ein Register und was ist seine Aufgabe ?

F 7.2 Was ist die ALU, was ist ihre Aufgabe ?

F 7.3 Welche Gruppen von Anschlußleitungen (Pins) hat eine CPU ?

F 7.4 Wie ist ein Von–Neumann Rechner charakterisiert?

F 7.5 Welchen logischen Adreßraum hat ein Mikroprozessor mit 24–Bit–Adressen?

F 7.6 Welche Möglichkeiten zur Erhöhung der Leistungsfähigkeit eines modernen Mikroprozessors gibt es?

F 7.7 Beschreibe kurz die Eigenschaften von CISC– und RISC–Architekturen mit ihren Vor- und Nachteilen.

F 7.8 Was bedeutet virtuelle Speichertechnik ?

F 7.9 Was ist Pipelineverarbeitung ?

F 7.10 Was ist ein Cache ?

F 7.11 Aus welchen Funktionsblöcken besteht ein Mikroprozessor?

7.8 Antworten zu den Fragen und Übungen zu Kapitel 7

A 7.1 Ein Register ist ein prozessorinterner und damit sehr schneller Speicher. R. dienen der (Zwischen-) Speicherung von Adressen oder Daten. Spezielle R. sind z.B. der Programmzähler, der die Adresse des nächsten auszuführenden Befehls enthält und das Statusregister, das den aktuellen Prozessorzustand mit den Bedingungsbits (flags) darstellt.

A 7.2 Die ALU ist die Arithmetisch-logische Einheit. Sie führt die arithmetischen und logischen Operationen an den Operanden aus. Bei der Verknüpfung zweier Operanden steht der eine im Akkumulator, der andere in einem Register oder in einer Speicherzelle des Arbeitsspeichers.

A 7.3 Adreß-, Daten- und Steuerbus, Versorgungsspannung, Takt

A 7.4 Die Rechnerstruktur ist problemunabhängig. Zur Lösung eines konkreten Problems muß ein Programm eingegeben werden. Programme und Daten werden binär codiert und im selben Speicher gehalten. Die korrekte Zuordnung ist nur aus dem Zusammenhang möglich. Aufeinanderfolgende Programmbefehle werden in aufeinanderfolgenden Speicherzellen abgelegt. Das Ansprechen geschieht durch Erhöhung der Befehlsadresse im Befehlswerk um eins. Durch Sprungbefehle kann von der gespeicherten Programmreihenfolge abgewichen werden.

A 7.5 $2^{24} = 16$ MByte

A 7.6 Erweiterungen der bestehenden Architektur: größere Wortbreite, breitere Adressen, höhere Taktrate usw.; Übernahme von Techniken aus dem Großrechnerbereich: virtuelle Speichertechnik, Speicherverwaltungseinheiten, Multitasking, interne Warteschlangen und Caches.

A 7.7 CISC (complex instruction set computer): breites Spektrum von zum Teil komplexen Instruktionen (mehr als 200) mit unterschiedlichsten Adressierungsarten, implementiert in Mikro-code. Damit direkte Unterstützung von Funktionen höherer Programmiersprachen und Betriebssystemen; resultiert in einfacheren Compilern, kürzerem Programmcode. Nachteile: komplexe Befehle werden selten angewendet (80% der Anwendungen nutzen nur 20% der vorhandenen Befehle) und sind aufwendig in der Codierung und Decodierung, Chipherstellung kompliziert und teuer, da Funktionen und Unterstützungsmittel hardwaremäßig implementiert werden müssen.

RISC (reduced instruction set computer): wenige (etwa 100) einfache Befehle mit einfachem, festem Format gestattet Verzicht auf Mikroprogrammierung und erlaubt schnelle Dekodierung. Befehlsabarbeitung in einem Takt, Chipherstellung einfacher. Speicherkommunikation beschränkt auf Laden und Speichern, andere Manipulationen geschehen in den Registern (bis zu 1000); damit einfachere Adressierung. Nachteil: größerer Compilationsaufwand zur möglichst optimalen Ordnung der Befehle und zur Nutzung des Pipelinings.

A 7.8 Ein Benutzerprogramm kann den gesamten logischen Adreßraum benutzen. Liegt eine Speicheradresse außerhalb des tatsächlich existierenden (physikalischen) Speichers, so sorgt das Betriebssystem in Zusammenarbeit mit dem Prozessor und der Speicherverwaltung dafür, daß Speicherbereiche aus Hauptspeicher und externem Speicher ausgetauscht werden (swapping).

A 7.9 Befehlspipelining: Die verschiedenen Phasen der Befehlsausführung (Befehl lesen, Adreßberechnung, Operanden lesen, Ausführen) können von voneinander unabhängigen Funktionseinheiten bearbeitet werden. Für Einzelbefehle werden diese Phasen nacheinander durchlaufen, jedoch können mehrere Befehle in unterschiedlichen Phasen gleichzeitig in Arbeit sein.

A 7.10 Ein schneller Zwischenspeicher, der z.B. zwischen Platte und Arbeitsspeicher angeordnet ist. Z.B. bei einem Lesezugriff wird zunächst im schnelleren Cache nachgesehen. Falls der entsprechende Sektor dort vorhanden ist, wird er ohne Wartezeit und damit wesentlich schneller als von der Platte aus dem Cache kopiert, ansonsten muß der Sektor zunächst von der Platte in den Cache geladen werden.

A 7.11 Datenwerk, Adreßwerk, Steuerwerk, Bustreiber.

Das Datenwerk besteht aus der Arithmetisch-Logischen Einheit (ALU), in ihm werden Daten bearbeitet. Das Adreßwerk besteht aus dem Befehlszähler und den Adreßregistern, es dient der Adreßberechnung. Das Steuerwerk ist mit allen Bausteinen über Steuerleitungen verbunden. Es besteht aus dem Befehlsdekodierer und den Befehlsregistern und dient der Befehlsausführung. Der Bustreiber dient der Behandlung des Prozessorbusses.

Kapitel 8

Rechnerschnittstellen

Der Betrieb von modernen Rechnern ist ohne Peripheriegeräte, wie Drucker, Terminals, Maus, Festplatten, Floppylaufwerke, grafische Tabletts und Meßgeräte nicht mehr vorstellbar. In diesem Kapitel werden als Beispiel die beiden für diese Geräte weit verbreiteten Schnittstellen Centronics und V.24 behandelt. Schnittstellen ermöglichen die Kommunikation zwischen dem Rechner und den peripheren Geräten. Die Kommunikation besteht im wesentlichen aus der Übertragung von Daten über ein physikalisches Medium, welches abstrakt als Datenkanal bezeichnet wird. In einer Kommunikation können mehrere Datenkanäle benutzt werden. Die Schnittstellen sind Bindeglieder zwischen dem Computer und seiner Peripherie.

Die Schnittstelle an sich ist physikalisch immer zwei Mal vorhanden, im Rechner und im angeschlossenen Gerät, Abb. 8.1. Im Rechner erscheint eine solche Schnittstelle als Karte, die in den Steckplatz des Rechnerbus gesteckt wird (siehe Kapitel 6).

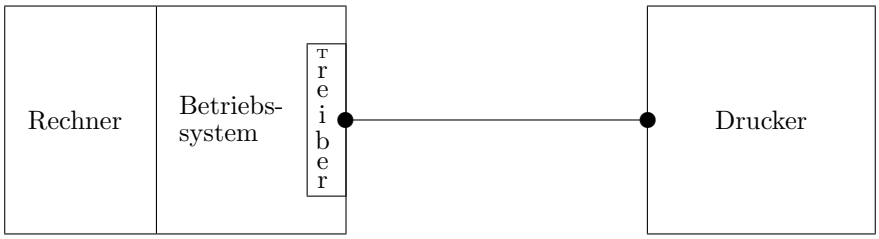


Abbildung 8.1: Schematische Darstellung einer Rechner–Drucker–Verbindung.

Die Steuerung der Kommunikation übernimmt ein spezielles Programm im Rechner, der Treiber, der im Betriebssystem des Computers eingebettet ist (siehe Kapitel 3). In den folgenden Kapiteln wird auf die Datenübertragung allgemein und mit konkreten Beispielen eingegangen. Kapitel 9 befaßt sich mit der erweiterten Datenkommunikation in Netzwerken.

8.1 Übertragung von Daten

Die Übertragung von Daten zwischen zwei Teilnehmern erfordert ein **Kommunikationsprotokoll**, d.h. eine Konvention, nach der die Kommunikation stattfindet. Die Kommunikationspartner „verabreden“ danach, wie der Aufbau, die Durchführung und der Abbau der Übertragungsverbindung stattfindet.

Neben dem Ablauf der Kommunikation beinhaltet die Konvention auch Informationen über den Status der Kommunikation.

Grundlage ist die Festlegung der gemeinsamen Sprache. Die Übertragungssprache im Computerbereich ist weitgehend eine Ansammlung von digital dargestellten alphanumerischen Zeichen sowie Sonderzeichen. Für die Darstellung der Zeichen hat sich der ASCII (ISO–7–Bit–Code) durchgesetzt. Er dient der Darstellung aller alphanumerischen Zeichen und Steuerzeichen im 7–Bit–Format.

In den folgenden Abschnitten wird auf die Betriebsart von Schnittstellen, die parallele und serielle Übertragung und die Steuerung der Übertragung eingegangen.

8.1.1 Betriebsarten

Die Betriebsarten sind durch zwei Modi gekennzeichnet, den Simplex– und den Duplex–Betrieb.

Im **Simplex–Betrieb** werden Daten nur in einer Richtung vom Sender zum Empfänger gesendet, so daß kein gegenseitiger Datenaustausch erfolgt. Für diese Art der Übertragung reicht ein Datenkanal aus.

Im **Halb–Duplex–Betrieb** werden Daten zwischen zwei Sende/Empfangsstationen ausgetauscht. Jede Station kann sowohl Daten empfangen als auch senden. Die Übertragung erfolgt jedoch nicht gleichzeitig in beide Richtungen, sondern es kann immer nur eine senden und die andere empfangen. Erst nach Ende des Sendevorgangs kann die Übertragungsrichtung geändert werden. Für diese Art der Übertragung reicht ein Datenkanal aus.

Im **Voll–Duplex–Betrieb** können die Daten auch zwischen zwei Sende/Empfangsstationen ausgetauscht werden, jedoch mit dem Unterschied, daß jede Station gleichzeitig empfangen und senden kann. Hier benötigt man zwei Datenkanäle zur Übertragung.

8.1.2 Serielle und parallele Übertragung

Grundsätzlich unterscheidet man bei der Datenübertragung zwischen dem seriellen und parallelen Prinzip. Seriell heißt, daß Informationen Bit für Bit hintereinander über einen Datenkanal gesendet werden. Parallel bedeutet, daß eine bestimmte Anzahl von Datenkanälen gleichzeitig zur Übertragung genutzt werden.

Wenn Zeichen, z.B. ASCII, übertragen werden, bedeutet eine serielle Übertragung, daß die einzelnen Bits des Zeichens bitseriell über einen Datenkanal gesendet werden. Werden ganze Bytes von Zeichen parallel über 8 Leitungen übertragen, die Zeichen aber hintereinander gesendet, spricht man von einer byteseriellen Übertragung. Die byteserielle Übertragung ist ein Spezialfall von serieller und paralleler Übertragung; sie wird beim IEC–Bus (siehe Abschnitt 8.4) angewandt.

Die serielle Übertragung ist weit verbreitet, da man im Prinzip nur ein Leitungspaar (Signal- und Rück- oder Masseleitung) benötigt, denn es wird jeweils nur ein Datenbit zur Zeit übertragen. Große Entfernungen lassen sich somit kostengünstig überbrücken, wogegen sich bei paralleler Übertragung die Kosten für das Kabel mit jedem weiteren Datenkanal erhöhen. Da die einzelnen Bits nacheinander über die Leitung geschickt werden, ist die serielle Übertragung relativ langsam.

Serielles Datenformat: Die Interpretation der empfangenen Daten stellt ein weiteres Problem der Datenkommunikation dar. Um die Zeichen korrekt zu empfangen und zu interpretieren, entstanden Synchronisationsverfahren zur zeitlichen Abstimmung, die auf einem bestimmten Datenformat basieren. Das Datenformat gibt die Form an, in der Zeichen gesendet werden. Es gibt das asynchrone und das synchrone Datenformat. Die Daten werden auf der Leitung durch einen definierten gültigen Spannungswert (Pegel) festgelegt, dessen Zeitdauer die Übertragungsgeschwindigkeit bestimmt. Diese muß auf Empfänger- und Senderseite gleich sein.

Asynchrones Datenformat: Dabei wird ein Zeichen, das übertragen werden soll, in ein Startbit und ein bis zwei Stopbits eingebettet. Das Startbit gibt den Beginn eines Zeichens an; die Stopbits geben das Ende eines Zeichens an, und geben dem Empfänger die Möglichkeit, sich auf das nächste Zeichen einzustellen. Dieses Format unterstützt das zeichenweise Senden (siehe auch Abschnitt 8.3).

Synchrones Datenformat: Dieses Format faßt mehrere Zeichen zu einem Block zusammen. Ein Block beginnt immer mit einem Synchronisationsbyte und endet mit einem speziellen Endezeichen. Die einzelnen Bytes werden ohne weitere Kennung übertragen. Zur Synchronisation wird parallel zum Block ein Takt gesendet, der ein Zeitraster vorgibt, das die Daten als gültig identifiziert.

Steuerung der Datenübertragung: Neben der Synchronisation der Übertragung von Zeichen sind auch Protokolle notwendig, die den Zustand und den Ablauf des Übertragungsvorgangs steuern. Die Protokolle starten und beenden eine Übertragung, legen die Richtung fest und geben Fehler bekannt. Für ein Übertragungsprotokoll sind Steuerzeichen notwendig.

ASCII–Steuerzeichen: Bestimmte Zeichen des ASCII werden in Übertragungsprotokollen benutzt. Die Steuerzeichen des ASCII–Zeichensatzes lassen sich in folgende Klassen aufteilen:

- Übertragung: ACK (Acknowledge; Bestätigung) bestätigt den fehlerfreien Empfang. EOT (End Of Transmission) zeigt das Ende der Übertragung an.
- Formatierung: Sie steuern die Form der Darstellung beim Empfänger. LF (Linefeed) bewirkt einen Zeilenvorschub. CR (Carriage Return) setzt wieder zum Zeilenanfang zurück. HT (Horizontal Tabulator) bewegt an die nächste Tabulatormarke, falls gesetzt.
- Gerätesteuerung: Steuerzeichen DC1, ..., DC4 werden je nach Endgerät verschieden interpretiert.
- Codeerweiterung: Bildung von zusätzlichen Schriftzeichen und Steuerbefehlen. SO (Shift Out) und SI (Shift In) für Groß- und Kleinschreibung, ESC (Escape) zur Definition von Steuerzeichen (Escape-Sequenzen).
- Sonstiges: Akustisches Signal BEL (Bell) und DEL (Delete) zum Löschen des vorherigen Zeichens.

Handshake–Verfahren: Zur Kontrolle der Datenübertragung zwischen zwei Geräten setzt man Handshake–Verfahren ein. Der Hauptgrund für solche Handshakes ist die oft unterschiedliche Verarbeitungsgeschwindigkeit von z.B. Rechner und Endgerät. Damit können sich die Geräte gegenseitig Bereitschaft zum Datenaustausch signalisieren.

- Hardwarehandshake: Elektrische Signale zwischen den Geräten steuern den Datenfluß. Die Signale sind für Schnittstellen vorgegeben, wobei bestimmte Signalleitungen benutzt werden.
- Softwarehandshake: Ein Programm auf beiden Schnittstellen kontrolliert die Datenübertragung. Ein Beispiel ist das XON/XOFF–Protokoll einer V.24–Schnittstelle.

8.2 Die Centronics–Schnittstelle

Diese Schnittstelle wurde von der Firma Centronics als Druckerschnittstelle entwickelt. Sie hat sich im PC–Bereich durchgesetzt und bildet einen Industriestandard. Sie überträgt die Daten parallel und asynchron im Simplex–Betrieb.

Funktionale Eigenschaften: Der Stecker verfügt über 36 Pole, wie an den Druckern zu finden, an den Rechnern findet man die 25polige Subminiatur–D–Buchse. Dabei sind jedoch keine Signale verloren gegangen, es wurde lediglich auf einige Masseleitungen verzichtet. Es werden nun die wichtigsten Leitungen der Centronics–Schnittstelle, gegliedert nach funktionalen Gesichtspunkten, vorgestellt. Die Leitungsgruppen sind in Abb. 8.2 für beide Stecker dargestellt.

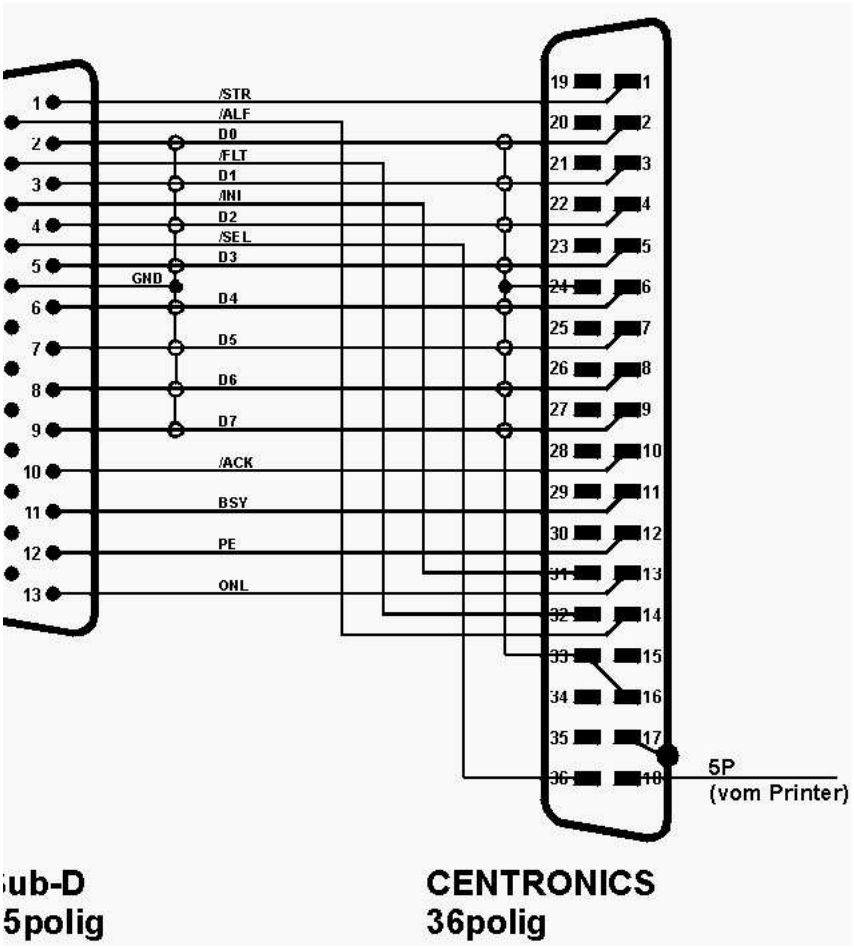


Abbildung 8.2: Leitungsgruppen der Centronics–Schnittstelle auf der Anschlußseite des 36–poligen Steckers und des 25–poligen Subminiatur–Steckers.

1. **Datenleitungen:** 8 Daten- (DATA 0. . . 7 oder DATA 1. . . 8) und 8 parallele Masseleitungen (GND; Ground). Die Daten- und Masseleitungen können als Twisted Pair (Verdrillte Leitungen; sie dienen zum Schutz vor Störsignalen) im Kabel ausgeführt werden.

2. **Steuerleitungen:** 3 Steuer- und 3 Masseleitungen. STROBE ist das Auftaktsignal vom Rechner an den Drucker zum Übertragen von Daten. ACKNLG (Acknowledge) ist das Quittungssignal des Druckers und meldet, daß er Daten übernehmen kann. BUSY zeigt an, daß der Drucker keine Daten empfangen kann, weil er z.B. mit Drucken beschäftigt ist.

3. **Meldeleitungen:** 2 Meldeleitungen. PE (Paper End) zur Anzeige des Papierendes und FAULT als Fehlermeldung.

Handshakeverfahren: Die Centronics–Schnittstelle arbeitet mit Hardware–Handshake, siehe Abb. 8.3. Die Datenübertragung beginnt mit dem Setzen des Übernahmesignals (STROBE) durch den Computer, was signalisiert, daß das Byte an den Datenleitungen anliegt. Der Drucker gibt durch das Quittungssignal (ACKNLG) die Bereitschaft bekannt, Daten zu übernehmen. Beginnt der Drucker mit dem Ausdrucken und kann deshalb keine Daten mehr empfangen, so setzt er das BUSY–Signal. Danach ist alles von den Möglichkeiten des Druckers abhängig, d.h. von der Größe des Druckerpuffers und von der Art des Druckers. Ein Nadeldrucker mit geringem Puffer wird z.B. nach dem Empfang des Formatzeichens CR das BUSY–Signal setzen und die Zeile drucken. Ein Laserdrucker dagegen wird warten, bis sein Hauptspeicher

von mehreren Megabyte voll ist und dann erst das BUSY–Signal setzen. Danach setzt der Drucker wieder ACKNLG und empfängt weiter Daten.

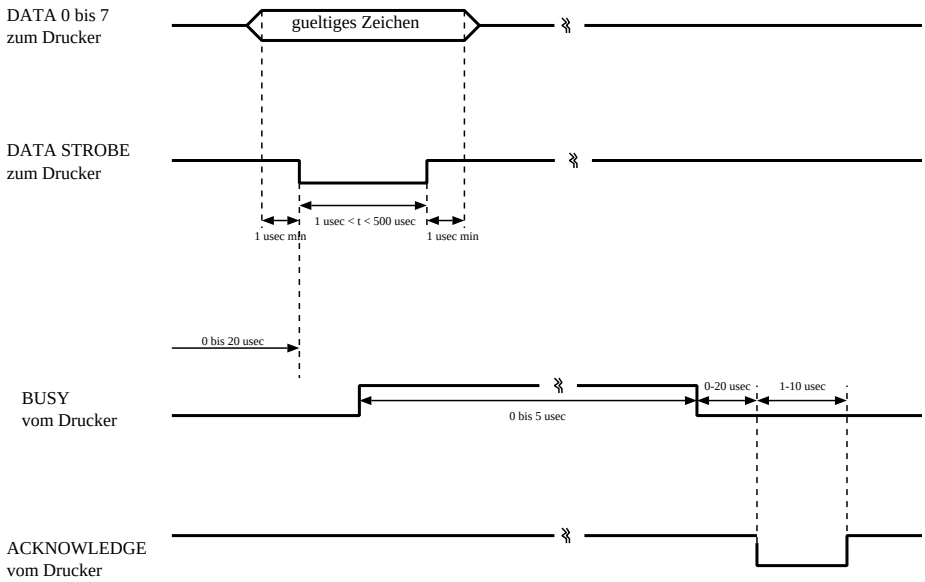


Abbildung 8.3: Centronics–Timing mit BUSY nach c't 1/84.

Die Kabellängen der Centronics–Schnittstelle liegen im Bereich von ein bis drei Metern und maximal von 8 Metern. Die Übertragungsgeschwindigkeit liegt bei 1000 Zeichen pro Sekunde. Weiteres entnehme man der Literatur (z.B. [c't 1/84] , [c't 10/86] , [c't 8/90]).

8.3 Die V.24–Schnittstelle

Um den Namen V.24 gibt es immer ein paar Verunsicherungen, da auch der Name RS–232–C verwendet wird, der jedoch etwas anderes bedeutet. V.24 ist eine Norm der CCITT (Comité Consultatif International Télégraphique et Téléfonique) und RS–232–C ist eine amerikanische Norm und entspricht der ISO–Norm 2110 und der DIN 60020. Die RS–232–C–Norm beschreibt neben dem funktionellen Ablauf auch die elektrischen Eigenschaften, V.24 standardisiert nur den funktionellen Ablauf einer asynchronen bitseriellen Übertragung sowie deren Bepinnung. Die elektrischen Anforderungen sind in der V.28–Norm festgelegt, so daß erst V.24 und V.28 der RS–232–C entsprechen. Wir wollen in diesem Abschnitt die V.24–Norm kurz erläutern.

Die Übertragung mit einer V.24–Schnittstelle erfolgt seriell im asynchronen Datenformat (siehe nächster Abschnitt) und kann auf Halb- oder Vollduplex eingestellt werden. Zunächst wollen wir den Einsatzbereich der V.24–Schnittstelle beschreiben. Der wichtigste Einsatz liegt im Bereich der Terminals, der Computer $\leftrightarrow$ Computer–Kommunikation, Modems, Drucker, Mäuse und grafischen Tabletts. Die Konfiguration von V.24–Schnittstellen ist wegen der Vielfalt der Einstellungsmöglichkeiten und der unterschiedlichen Formate und Verbindungsmöglichkeiten der Schnittstelle nicht einfach.

Mit der **asynchronen Übertragung** eines Zeichens wird, ausgehend von einem Ruhezustand, zu irgendeinem beliebigen Zeitpunkt begonnen. Die Norm definiert als Ruhezustand den MARK–Zustand (logisch '1') der Sender–Ausgangsleitung. Dieser Zustand liegt nach dem Einschalten vor und wird nach jeder Übertragung wieder eingenommen. Bevor das erste Bit eines Zeichens ausgegeben werden darf, ist der Empfänger in geeigneter Weise vorzuwarnen, da dieser, falls das erste Bit den Wert '1' hat, keinen Unterschied zum Ruhezustand bemerken würde. Dazu dient das Startbit: für die Dauer eines Bits gibt der

Sender SPACE aus (logisch '0'), was sich eindeutig vom Ruhezustand unterscheidet und dem Empfänger Gelegenheit gibt, seine Abfragelogik auf das Einlesen der Datenbits vorzubereiten. Die Datenbits, z.B. 7–Bit–ASCII–Zeichen, folgen unmittelbar auf das Startbit, mit dem LSB voran (engl. **l***east* **s***ignificant* **b***it* = Bit 1). Nach dem letzten Bit, dem MSB (engl. **m***ost* **s***ignificant* **b***it* = Bit 7), kann noch ein Paritätsbit zur Erkennung von Übertragungsfehlern eingefügt werden. Man unterscheidet zwischen gerader (even) und ungerader (odd) Parität. Gerade Parität bedeutet zum Beispiel, daß die Anzahl der übertragenen gesetzten Bits ('1') einschließlich Paritätsbit gerade ist. Der Sender setzt also das Paritätsbit, wenn die Anzahl der gesetzten Bits im Zeichen ungerade ist. Der Empfänger untersucht den Bitstrom nach der gleichen Vorschrift und kann so Übertragungsfehler feststellen. Das Ende der Übertragung eines Zeichens kennzeichnet ein Stopbit (MARK), das 1, 1,5 oder 2 Bitzeiten dauern kann (daher 1, 1,5 oder 2 Stopbits). Ein Beispiel für eine serielle asynchrone Übertragung ist in Abb. 8.4 dargestellt.

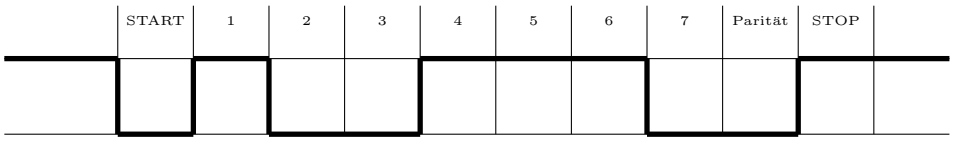


Abbildung 8.4: Beispiel für eine serielle asynchrone Übertragung der ASCII–Ziffer 9 (39_{16}) mit Startbit, einem Stopbit und gerader Parität. (nach c't 12/86)

Funktionale Eigenschaften: Der Stecker verfügt über 25 Pole, wovon aber nicht alle belegt sind. Trotz aller Normen gibt es heute immer noch Rechnerhersteller, die sich nicht an die Normen halten und andere Belegungen benutzen (z.B. IBM PCs und Kompatible). Die Belegung (Lötseite) des normgerechten 25–poligen Steckers und der platzsparenden 9–pol–Verbindung in der Mindestausstattung für bidirektionalen Transfer sind in Abb. 8.5 dargestellt.

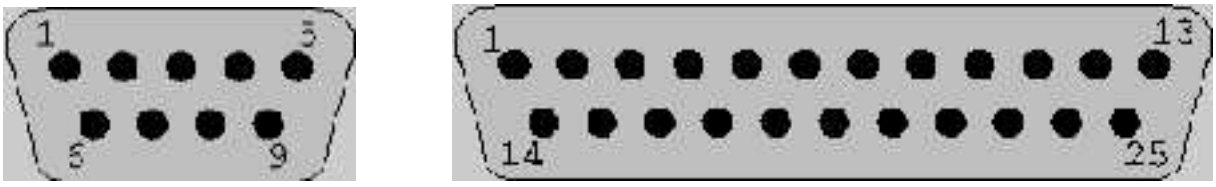


Abbildung 8.5: Normstecker 25–polig(rechts), 9–pol–Ausführung (links). Mindestausstattung für bidirektionalen Transfer (Dreileitungsverbindung).

Pin	Description	Pin	Description	Pin	Description
1	Earth Ground	10	Reserved	18	Unassigned
2	TXD – Transmitted Data	11	Unassigned	19	Secondary RTS
3	RXD – Received Data	12	Secondary DCD	20	DTR – Data Terminal Ready
4	RTS – Request To Send	13	Secondary CTS	21	Signal Quality Detect
5	CTS – Clear To Send	14	Secondary TXD	22	Ring Detect
6	DSR – Data Set Ready	15	Transmit Clock	23	Data Rate Select
7	GND – Logic Ground	16	Secondary RXD	24	Transmit Clock
8	DCD – Data Carrier Detect	17	Receiver Clock	25	Unassigned
9	Reserved				

Tabelle 8.1: Belegung der Pins des 25–pol–Steckers

Die einschlägigen Normen gehen von einer bestimmten Verbindungs–Philosophie aus, bei der zwei Kategorien von Geräten benötigt werden: DTE (Data Terminal Equipment; Datenendeinrichtung/en) und

Pin	Description	Pin	Description
1	DCD – Data Carrier Detect	6	Data Set Ready
2	RXD – Received Data	7	RTS – Request To Send
3	TXD – Transmitted Data	8	CTS – Clear To Send
4	DTR – Data Terminal Ready	9	Ring Detect
5	GND – Logic Ground		

Tabelle 8.2: Belegung der Pins des 9–pol–Steckers

DCE (Data Communication Equipment; Datenübertragungseinrichtung/en). Ein DTE ist ein Gerät, welches am Anfang oder am Ende einer Datenübertragungskette steht und Daten sendet bzw. empfängt. Bei einem DCE handelt es sich um eine Art Zwischenstation, die zwischen DTE und der Übertragungsstrecke vermittelt, wenn beide mit unterschiedlichen Darstellungen der zu übertragenden Information arbeiten; die Daten selbst bleiben dabei unverändert. Ein Beispiel für ein DCE ist das Modem, und typische DTE's sind Computer, Terminals usw. Die Unterscheidung zwischen DTE und DCE ist für V.24 besonders wichtig.

Die Leitungen der Schnittstelle werden nun, nach funktionalen Gesichtspunkten gegliedert, dargestellt.

- Datenleitungen: Pin2 (TxD, Transmitted Data) Sendeleitung und Pin3 (RxD, Received Data) Empfangsleitung.
- Steuerleitungen: Pin4 (RTS, Request To Send) Einschalten des Senders und Pin20 (DTR, Data Terminal Ready) zeigt die Betriebsbereitschaft des Endgerätes an; es kann als Busy–Signal verwendet werden.
- Meldeleitungen: Pin5 (CTS, Clear to Send) Bereitschaft zum Senden, Pin6 (DSR, Data Set Ready) zeigt die Betriebsbereitschaft an und Pin8 (DCD, Data Carrier Detect) leitet den Signalpegel weiter.
- Taktleitungen: Pin15 (TxC, Transmitter Signal Element Timing) gibt den Sendetakt von der DCE an, Pin17 RxC (Receiver Signal Element Timing) Empfangstakt von der DCE und Pin24 (Transmitter Signal Element Timing) Sendetakt zur DCE.

Die Kabellänge bei V.24–Schnittstellen kann erheblich länger sein als bei Centronics–Schnittstellen. Sie wird durch die Übertragungsgeschwindigkeit bestimmt. Je höher die Übertragungsgeschwindigkeit, desto kürzer ist die Kabellänge. Als Richtwert gilt bei einer Übertragungsgeschwindigkeit (siehe unten) von weniger als 1000 bit/s eine Kabellänge bis zu 1000 m. Es gibt auch noch eine andere Einheit für die Übertragungsgeschwindigkeit: Baud. Baud gibt an wieviel Übertragungseinheiten pro Sekude übertragen werden können. Werden also pro Einheit 2 Bit übertragen gilt: 9600 Baud = 19200 bit/s.

Verbindungen von V.24–Schnittstellen: Bei der Verbindung von V.24–Schnittstellen wird DCE und DTE benutzt. Es werden die wichtigsten Kombinationen von Verbindungen vorgestellt.

Die einfachste Verbindung ist die **Zweidrahtverbindung** mit den Pins 2, 3 und 7. Im Falle einer Kommunikation zwischen einem DCE und einem DTE werden die Leitungen TxD und RxD direkt zwischen den Teilnehmern verbunden. Im Falle einer Kommunikation zwischen zwei DTE werden RxD und TxD gekreuzt. Diese Übertragungsart ist die Minimalform, mit der diese Schnittstelle betrieben werden kann. Um die Sicherheit einer solchen Übertragung zu gewährleisten, setzt man Softwareprotokolle ein.

Mehrdrahtverbindungen: Um eine Kontrolle durch Hardwarehandshake zu erreichen, werden die anderen Leitungen der Schnittstelle benutzt. Eine einfache Hardware–Handshake–Lösung ist die Hinzunahme der Leitung 20 (DTR). Sie meldet die Bereitschaft, weitere Daten zu empfangen, und stellt nun ein Ready/Busy–Signal dar, daher auch die Bezeichnung Ready/Busy–Protokoll.

Eine andere Erweiterung bietet die Integration der Signale 4 (RTS) und 5 (CTS). Über CTS teilt der Sender mit, daß weitere Daten vorhanden sind. Mit RTS fordert der Empfänger den Sender auf, mit der Übertragung zu beginnen. Die Übertragung findet nur statt, wenn beide Signale gesetzt sind. Die Leitungen werden äquivalent zur Zweidrahtverbindung parallel (DCE) oder gekreuzt (DTE) durchgeschleift.

Die letzte weit verbreitete Kombination beinhaltet die zusätzliche Benutzung der Signale 6 (DSR) und 20 (DTR). Diese Leitungen zeigen zusätzlich die Betriebsbereitschaft der Teilnehmer an; sie müssen zur Übertragung beide aktiv sein.

Es gibt zu den hier vorgestellten Verbindungen noch viele weitere, auf die wir aber nicht mehr eingehen wollen.

Die **Übertragungsgeschwindigkeit** einer V.24–Schnittstelle (Baudrate) läßt sich in festen Schritten einstellen. Einen Überblick gibt die Tabelle 8.3.

110	150
200	300
600	1200
1800	2400
4800	9600
19200	38400

Tabelle 8.3: Die Standard–Übertragungsgeschwindigkeiten in Baud

Aus der Baudrate kann man die Zeichenrate ableiten, also die Anzahl der übertragenen Zeichen je Zeiteinheit. So werden beispielsweise bei einem Datenformat von einem Startbit, 7 Datenbit und zwei Stopbits und einer Geschwindigkeit von 300 Baud (für Akustikkoppler) genau 30 Zeichen pro Sekunde übertragen.

Das Datenformat der V.24–Schnittstelle ist bereits weiter oben beschrieben worden.

Das wichtigste **Protokoll** ist das XON/XOFF–Protokoll. Es ist ein empfängergesteuertes Software–Handshaking, welches ähnlich dem Ready/Busy–Protokoll funktioniert: Das empfangende Gerät (z.B. der Drucker) nimmt solange Daten entgegen, bis sein Puffer voll ist, und schickt dann über seine Sendeleitung ein XOFF–Zeichen (ASCII: DC3 = 13_{16} , Control–S) zum Sender (z.B. der Computer). Sobald dieser das Zeichen empfängt, geht er in einen Wartezustand. Wenn der Empfänger wieder Daten entgegennehmen kann (z.B. weil der Druckerpuffer geleert ist), sendet er das XON–Zeichen (ASCII: DC1 = 11_{16}, Control–Q), und der Sender fährt an der Stelle fort, an der er angehalten hatte. Übrigens ist das Verfahren, eine laufende Bildschirmausgabe mit Control–S anzuhalten und mit Control–Q fortzusetzen, nichts anderes, als eine Anwendung des XON/XOFF–Protokolls.

Für das Software–Handshaking braucht man also neben den Datenleitungen (Pins 2 und 3) und Masse (Pin 7) keine weiteren Verbindungen zwischen DTE und DCE herzustellen, was wenig aufwendige Kabelverbindungen ermöglicht.

Wer noch tiefer in die Funktion serieller Schnittstellen einsteigen will, sei abschließend auf die Literatur zu diesem Thema verwiesen (z.B. [c't 12/83] , [c't 12/86] , [c't 8/90]).

8.4 Weitere Schnittstellen: Der IEC–Bus

Die IEC–Schnittstelle nimmt unter den bekannten Schnittstellen eine Sonderstellung ein. Sie ist zwar die aufwendigste und komplizierteste, aber die genormtem Eigenschaften werden von fast allen Herstellern so exakt eingehalten, wie bei keiner anderen Schnittstelle. Auch können mehrere Geräte einfach parallel geschaltet werden (Busschnittstelle).

Der IEC–Bus ist ein Bus für Meßsysteme und Peripheriegeräte. Die Entwicklung des Bussystems wurde schon 1965 von der Firma Hewlett Packard vorgenommen und existiert dort unter dem Namen HP–IB (Hewlett Packard Interface Bus). Erst viel später wurde dieser Bus wurde von IEEE als IEEE–488 und von der DIN als DIN IEC–625 genormt. Das Bussystem stellt im Meßgerätebereich einen Industriestandard dar.

Die Entwicklung wurde geleitet vom Wunsch nach einem Bussystem, das die unterschiedlichsten Meß- und Peripheriegeräte mit dem Computer verbindet und ein hohes Maß an Übertragungssicherheit, sowie eine dynamische Konfiguration des Bussystems zur Laufzeit gewährleistet. Ein weiterer Gesichtspunkt war, daß Geräte mit unterschiedlicher Verarbeitungsgeschwindigkeit während des Betriebes an den Bus angeschlossen und auch abgeklemmt werden können, und zwar so, daß das System sich darauf dynamisch einstellen kann. Die unterschiedlichen Verarbeitungsgeschwindigkeiten werden dadurch aufgefangen, daß das System sich auf das langsamste am Bus angeschlossene Gerät einstellt.

8.4.1 Funktionsweise des IEC–Busses

Der IEC–Bus unterscheidet Geräte mit unterschiedlichen Rollen. Es gibt für die Geräte drei Rollen, die sie einnehmen können:

Steuereinheit	—	Controller
Sprecher	—	Talker
Empfänger	—	Listener

Damit ein IEC–Bus funktioniert, müssen mindestens zwei Geräte am Bus angeschlossen sein. Die Geräte am IEC–Bus können so gebaut sein, daß sie eine, zwei oder drei Rollen übernehmen können.

Talker (sendet Daten): Höchstens ein Talker darf zu einer Zeit aktiv sein. Ein Talker wird aktiv, wenn der Controller das Gerät als Talker adressiert hat.

Listener (empfängt Daten): Mehrere Listener dürfen gleichzeitig aktiv sein. Das Gerät wird als Listener aktiv, wenn der Controller das Gerät als Listener adressiert hat.

Controller: Höchstens ein Controller darf zu einer Zeit aktiv sein. Von mehreren angeschlossenen Controllern ist einer als system controler ausgezeichnet. Der Controller steuert den Bus, kann aber in dieser Eigenschaft Gerätenachrichten weder senden noch empfangen.

Die Kontrolle über den Bus hat der Controller, der meistens der Computer ist. Er kann auf dem Bus senden, empfangen und ihn steuern. Daher kommen auch die Begriffe sprechen (talking), hören (listening) und steuern (controling).

Ein Beispiel für einen Talker/Listener ist ein Digitalvoltmeter, das sowohl Einstelldaten empfängt, aber auch die gemessenen Werte zurückschickt. Ein Beispiel für einen Listener ist ein Drucker, der nur Daten empfangen kann und für Meßprotokollausdrucke verwendet werden kann. Ein reiner Talker kann z.B. ein Temperaturfühler sein, der Daten, die er gemessen hat, verschickt.

Das Senden von Daten auf den Bus kann bei mehreren Sprechern nur nacheinander erfolgen. Die Koordination der Busteilnehmer übernimmt der Controller. Er bestimmt, wer auf den Bus zugreifen und welches Gerät in welchem Modus, Talker oder Listener, arbeiten darf. Zur Umsetzung seiner Arbeit steht ihm dabei ein umfangreicher Befehlssatz zur Verfügung, auf den wir aber hier nicht weiter eingehen.

8.4.2 Funktionale Eigenschaften

Der Bus besteht aus 16 Leitungen, die hier nach funktionalen Gesichtspunkten gegliedert angegeben werden.

- **Die Datenleitungen:** Der IEC–Bus besteht aus 8 parallelen Datenleitungen. Sie dienen nicht nur der Übertragung von Daten (z.B. Meßwerte), sondern auch der Übertragung von Geräteadressen und Statusinformationen.

- **Die Handshakeleitungen:** 3 Handshakeleitungen. DAV (Data Valid): gültige Daten stehen auf dem Bus bereit. NRFD (Not Ready For Data): keine Bereitschaft zur Übernahme der Daten. NDAC (No Data Accepted): es wurden keine Daten empfangen; dies entspricht einem Acknowledge.

- **Steuerleitungen:** 5 Steuerleitungen. REN (Remote Enable): Signal zur Fernsteuerung von Geräten. EOI (End Or Identify): Endesignal für das letzte Byte des Datenblocks oder Identifizierungssignal zur Bekanntgabe einer Datenanforderung der Geräte. IFC (Interface Clear): das Interface aller Geräte wird initialisiert. SRQ (Service Request): Anforderungen einer Bedienung des Geräts von der Steuereinheit. ATN (Attention): Achtungssignal bei der Übertragung von Statusinformationen oder Adressen.

8.4.3 Handshakeverfahren

Die Übertragung von Daten erfolgt ähnlich der Centronics–Schnittstelle mit Hilfe der Handshakeleitungen. Wenn alle Empfangsgeräte ihre Empfangsbereitschaft durch das Signal NRFD bei zurückgenommenem Signal NDAC anzeigen, kann die Übertragung beginnen. Nun legt der Sender die Daten auf den Datenbus und erklärt sie mit DAV für gültig. Die Empfänger übernehmen die Daten und erklären danach, daß sie keine Daten mehr empfangen können, indem sie das NRFD Signal zurücknehmen. War die Datenübernahme der Empfänger erfolgreich, so setzen sie das Signal NDAC für den Sender. Dieser weiß, daß die Daten angekommen sind und nimmt die Daten vom Bus, indem er das Signal DAV zurücksetzt. Die Empfänger setzen das Signal NRFD, um zu zeigen, daß sie die Datenübernahme abgeschlossen haben.

Die funktionalen Zusammenhänge der IEC–Schnittstelle sind komplex. Weitere Details sprengen den Rahmen einer Einführung, sie können in der Literatur nachgelesen werden (z.B. [c't 2/84]). [1]

8.5 Zusammenfassung

Schnittstellen ermöglichen die Kommunikation zwischen Rechner und peripheren Geräten. Sie bestehen aus der physikalischen Schnittstelle (Hardware) und werden von einem Treiberprogramm gesteuert. Eigenschaften und Funktion der Schnittstellen entsprechen meist gängigen Standards (Normen oder de facto–Standards). Aufbau, Durchführung und Beenden der Kommunikation erfolgen nach einem Kommunikationsprotokoll, das auch Statusinformationen beinhaltet.

Schnittstellen werden im Simplex (S)–, Halb–Duplex (HD)– oder Voll–Duplex (VD)–Modus betrieben. Dabei ist die Datenübertragung nur in einer Richtung (S) bzw. in beiden Richtungen, wechselseitig (HD) oder gleichzeitig (VD) möglich. Informationen werden dabei seriell, d.h. Bit für Bit über einen Datenkanal, oder parallel, d.h. mehrere Bits gleichzeitig (parallel) über eine entsprechende Anzahl von Datenkanälen übertragen.

Synchronisationsverfahren sichern richtigen Empfang und Interpretation von Daten mithilfe des synchronen oder asynchronen Datenformats. Kommunikationsprotokolle benutzen zur Steuerung der Datenübertragung Steuer–Zeichen (Software–Handshake) oder Signal–Leitungen (Hardware–Handshake).

Für den Anschluß von Druckern hat sich die Centronics–Schnittstelle durchgesetzt. Dieser Industrie–Standard überträgt Daten parallel und asynchron im Simplex–Modus und arbeitet mit Hardware–Handshake.

[1] Dieses Kapitel basiert auf dem entsprechenden Kapitel des Skripts von Prof. Biedl, FB 13.
W. W. Baumann, Januar 1991

Die Schnittstelle nach der V.24–Norm dient der seriellen, asynchronen Übertragung. Dabei werden die Bits eines Zeichens, evt. incl. Paritätsbit zur Übertragungskontrolle, in Start- und Stop–Bit(s) eingebettet. Zur Übertragungssteuerung wird oft das XON/XOFF–Protokoll (Software–Handshake) eingesetzt.

Der IEC–Bus ist die Standard–Schnittstelle für Meßsysteme. Geräte können als Sender (Talker) oder Empfänger (Listener) arbeiten, der Bus wird von einer Steuereinheit (Controller) gesteuert. Das Handshake–Verfahren ist ähnlich dem der Centronics–Schnittstelle.

8.6 Fragen und Übungen zu Kapitel 8

F 8.1 Was ist der Unterschied zwischen serieller und paralleler Übertragung?

F 8.2 Was ist die Centronics–Schnittstelle? Wofür wird sie typischerweise eingesetzt? Wie arbeitet sie?

F 8.3 Was ist die V.24 Schnittstelle? Wofür wird sie typischerweise eingesetzt? Unter welchen Namen ist sie noch bekannt?

F 8.4 Welche Schnittstelle wird hauptsächlich im Meßgerätebereich verwendet?

F 8.5 Erkläre den Unterschied zwischen synchroner und asynchroner Übertragung.

F 8.6 Erkläre kurz die Funktionsweise einer IEC–Schnittstelle.

F 8.7 Was versteht man unter Handshaking? Welche prinzipiellen Handshake–Arten gibt es?

8.7 Antworten zu den Fragen und Übungen zu Kapitel 8

A 8.1 Die serielle Übertragung benutzt einen Datenkanal, die parallele mehrere, d.h. die einzelnen Bits eines Bytes werden bei serieller Übertragung hintereinander über die gleiche Leitung geschickt, bei paralleler Übertragung werden sie gleichzeitig über eine entsprechende Anzahl paralleler Datenleitungen gesendet.

A 8.2 Die Centronics–Schnittstelle ist ein Industriestandard zum Anschluß von Druckern. Die Schnittstelle arbeitet parallel, asynchron und im Simplex–Betrieb.

A 8.3 Eine serielle Schnittstelle für Maus, Modem, Akkustikkoppler, Plotter und Drucker. RS–232 C

A 8.4 Der IEC–Bus ist im Meßgerätebereich weit verbreitet.

A 8.5 Die asynchrone Übertragung kann zu jeder beliebigen Zeit beginnen. Dabei wird ein Zeichen in ein Startbit und 1 oder 2 Stopbits eingebettet. Das Startbit zeigt den Beginn der Übertragung an. Das synchrone Verfahren sendet parallel zu den Daten einen Synchronisationstakt und faßt mehrere Zeichen zu einem Block zusammen, der mit einem Synchronisationsbyte beginnt und mit einem speziellen Endezeichen abgeschlossen wird.

A 8.6 Die IEC–Schnittstelle ist eine Busschnittstelle, d.h. alle an der Datenübertragung beteiligten Geräte sind parallel auf den Bus geschaltet. Die Geräte können verschiedene Funktionen haben: Talker (kann Daten senden), Listener (kann Daten empfangen) und Controller (steuert und überwacht das Geschehen auf dem Bus). Der Controller bestimmt, welches Gerät in welcher Funktion (Talker, nur einer zu jeder Zeit; Listener, auch mehrere gleichzeitig) auf dem Bus aktiv sein darf.

A 8.7 Handshake–Verfahren dienen zur Kontrolle der Daten-Übertragung zwischen zwei Geräten, die oft unterschiedliche Verarbeitungsgeschwindigkeit haben, z.B. Rechner und Drucker. Die Geräte signalisieren sich gegenseitig Bereitschaft zum Datenaustausch.

Beim Hardware–Handshake steuern elektrische Signale auf den Signalleitungen zwischen den Geräten den Datenfluß. Für Schnittstellen sind die Signale und die benutzten Signalleitungen vorgegeben. Beispiel: Centronics–Schnittstelle.

Beim Software–Handshake kontrolliert ein Programm auf beiden Schnittstellen die Datenübertragung. Die Signale werden als Steuerzeichen über die Sendeleitung geschickt. Beispiel: XON/XOFF–Protokoll einer V.24 Schnittstelle.

Kapitel 9

Netzwerke

9.1 Allgemeine Einführung

Was ist eigentlich ein Computer–Netzwerk?

Computer–Netzwerke sind in den letzten Jahren in starkem Maße populär geworden und haben sich in vielen Bereichen zu einem Standard entwickelt. Eine Definition von Tanenbaum [15] besagt: Ein Computer–Netzwerk ist eine Ansammlung zusammenhängender, selbständiger Computer.

Zusammenhängend bedeutet, dass die Computer Informationen über das Netzwerk austauschen können. **Selbständig** meint, dass kein Rechner den anderen unmittelbar steuert. Die über die Netzwerke ausgetauschten Informationen können Programme, Daten oder Nachrichten sein. Der Benutzer muss aber immer explizit Kommandos verwenden, um innerhalb eines Netzes bestimmte Aktionen zu starten. Im Lauf der Jahre haben sich Qualität und Anzahl der Programme für Computer–Netzwerke drastisch erhöht.

Ein Beispiel für ein Computer–Netzwerk sind mehrere durch ein Ethernet (kostengünstiges Netzwerk auf der Basis von Koaxialkabeln, siehe Abschnitt 9.6, verbundene Workstations oder Personal–Computer, die von einem Servicerechner (Server) mit den notwendigen Programmen und Daten über das Netzwerk versorgt werden, siehe unser Micropool.

Computer–Netzwerke haben wesentliche Vorteile.

1. Hardware–**Ressourcen** (Rechner, Festplatten, Drucker) und Software–Ressourcen (spezielle Anwendungsprogramme) können effektiver ausgenutzt werden. Das Netzwerk ermöglicht den Zugriff auf diese Ressourcen, unabhängig davon, wo sich Hard-/Software und Benutzer befinden, die Ressourcen können **zentral gewartet** und über das Netzwerk einer großen Zahl von Benutzern zugänglich gemacht werden.

2. Die **Zuverlässigkeit** eines Systems vernetzter Rechner ist höher als die eines einzelnen Rechners. Fällt in einem Rechnernetz ein Rechner aus, können Benutzer auf andere Rechnerkapazitäten ausweichen, während beim Ausfall eines stand–alone Rechners sämtliche Benutzer auf dem Trockenen sitzen. Die Verfügbarkeit von Software kann sich erhöhen, indem man Daten nicht nur auf einer Maschine hält, sondern auf mehrere Rechner verteilt. Fällt einer dieser Rechner aus, können die Daten der anderen Rechner benutzt werden. Dieser Aspekt von Rechnernetzen ist besonders wichtig in den Bereichen Militär, Prozesssteuerung, Flug- oder Bankwesen.

3. Das **Preis/Leistungsverhältnis** mehrerer kleiner, aber vernetzter Rechner ist günstiger als das eines großen Rechners. Großrechner sind von der Verarbeitungsgeschwindigkeit zwar vielleicht zehn

mal schneller als die schnellsten Mikroprozessoren, kosten jedoch vielleicht tausend mal mehr als eine Workstation (Arbeitsplatzrechner) oder ein Personal Computer. Das führte zur Entwicklung von Arbeitsplatzrechnern mit Netzwerkanschlüssen, die einem Benutzer recht große Rechenleistung zur Verfügung stellen, wobei die Programme und Daten auf einem oder mehreren Servern (File Server) gespeichert sind.

Auch wenn die Vernetzung von Rechnern viele Vorteile bringt, so gibt es trotzdem einige Nachteile, die hier zumindest kurz erwähnt werden sollen. Meist ist der Hauptgrund für eine Vernetzung Daten und Informationen auszutauschen und Dienste netzwerkweit anzubieten. Aber genau hieraus ergibt sich auch ein entscheidender Nachteil: die erstmal geringere Sicherheit der Daten vor unberechtigten Zugriffen. Zu einem unvernetzten Rechner benötigt ein Angreifer noch direkten physischen Zugang, wobei er bei vernetzten Rechnern aus der Ferne operieren kann, von einem Rechner im eigenen Teilnetz oder sogar einem Rechner im Internet aus. Die Administratoren von Rechnernetzwerken benötigen daher eine entsprechende Sensibilisierung für Sicherheitsprobleme, die bei einer Vernetzung unweigerlich auftreten.

Zudem sollte bedacht werden, dass der Aufbau eines Netzwerkes zusätzliche Hardware erfordert. Diese Hardwarekomponenten wie Netzwerkkarten, Kabel, Router, etc. können ausfallen und so mitunter alle am Netzwerk angeschlossenen Rechner funktionsunfähig werden lassen.

Zusammenfassend kann jedoch festgestellt werden, dass in der Praxis die Vorteile klar überwiegen und die vorhandenen Nachteile z.B. durch ein an die Erfordernisse angepasstes Sicherheitsdenken minimiert werden können.

Wir haben schon in Kapitel 3 verschiedene Anwendungsmöglichkeiten innerhalb von Netzwerken kennengelernt. Diese und weitere, neue Anwendungen beeinflussen nicht unbeträchtlich die gesellschaftliche Entwicklung.

Netzwerke werden durch ihre räumliche Ausdehnung unterschieden. Das Netzwerk unseres Institutes ist ein lokales Netzwerk, ein LAN (engl. **l***ocal* **a***rea* **n***etwork*). Im Gegensatz dazu gibt es die WAN's (engl. **w***ide* **a***rea* **n***etwork*). Eine Aufgliederung unter diesem Gesichtspunkt ist Tabelle 9.1 zu entnehmen.

Entfernung	**Ort der Prozessoren**	**Netzwerkart**
0,1 m	Rechnerplatine	Multiprozessorbord
1 m	Systemeinheit	Multiprozessorsystem
10 m	Raum	LAN
100 m	Gebäude	LAN
1 km	Campus	LAN
10 km	Stadt	WAN
100 km	Land	WAN
1000 km	Kontinent	Verbindung von WAN's

Tabelle 9.1: Kategorien von Netzen (nach der Entfernung eingeteilt)

9.2 Die Architektur von Computer–Netzwerken

Wichtig in einem Computer–Netzwerk ist nicht nur das Übertragungsmedium und der physikalische Anschluss, sondern vor allem die Software, die die Kommunikation steuert. Unter Architektur ist dann auch die Struktur der Kommunikationssoftware zu verstehen. Die Kommunikationssoftware hat unter anderem folgende Aufgaben zu bewältigen:

- Verbindung: Aufbauen, Aufrechterhalten und Abbauen von Verbindungen zwischen Rechnern.

- Wegewahl (engl. *routing*): Da nicht zwischen allen Rechnern eine direkte Verbindung besteht, muss eine geeignete indirekte Verbindung gefunden werden.
- Transparente Übertragung: Beliebige Bit- und Bytekombinationen müssen so übertragen werden, dass sie beim Empfänger unverändert ankommen.
- Flusskontrolle: Da die in einem Netz verbundenen Rechner unterschiedliche Übertragungsgeschwindigkeiten haben können, muss eine Überlastung eines empfangenden Rechners vermieden werden.
- Multiplexing: Falls über eine physikalische Leitung mehrere indirekte Verbindungen aufrechterhalten werden müssen, wird die physikalische Leitung diesen Verbindungen in bestimmten Zeitabschnitten zu Verfügung gestellt (zeitliche Aufteilung).
- Splitting/Recombining: Aus Effizienzgründen können Datenpakete aufgespaltet (engl. *splitting*), über verschiedene physikalische Leitungen versendet und beim Empfänger wieder in der richtigen Reihenfolge zusammengesetzt (engl. *recombining*) werden.
- Zuverlässigkeit: Übertragungsfehler kommen immer wieder vor. Um eine Verbindung absolut zuverlässig zu gestalten, müssen auftretende Fehler durch geeignete Maßnahmen korrigiert werden, z.B. durch Wiederholung des Sendens von Datenpaketen.

Dieser Katalog von Anforderungen an Kommunikationssoftware ist nur ein kleiner Ausschnitt, soll uns aber vorerst genügen. Da die Komplexität der Kommunikationssoftware sehr groß ist, wird das Problem softwaretechnisch in Schichten aufgeteilt, die streng hierarchisch aufeinander aufbauen. Die Anzahl der Schichten, der Name und die Funktion jeder Schicht unterscheiden sich von Netzwerk zu Netzwerk.

- Prima: Alle bekannten Netze legen eine Schichtenhierarchie zugrunde.
- Leider: Die Anzahl, Namen, Aufgaben und Funktionsweisen der einzelnen Schichten sind in allen Netzen unterschiedlich.
- Fazit: Internationale Standards sind notwendig.

Daher hat die ISO (International Standardization Organisation) ein Standardmodell für vorhandene und zukünftige Kommunikationssoftware entwickelt, das Reference Model of Open Systems Interconnection (OSI). Es legt sieben Kommunikationsschichten zugrunde, die in Tabelle 9.2 dargestellt sind. Das OSI–Modell wird in Abschnitt 9.4 genauer erläutert.

Schicht 7	Anwendungsschicht	Application Layer
Schicht 6	Darstellungsschicht	Presentation Layer
Schicht 5	Kommunikationssteuerungsschicht	Session Layer
Schicht 4	Transportschicht	Transport Layer
Schicht 3	Vermittlungsschicht	Network Layer
Schicht 2	Abschnittssicherungsschicht	Data Link Layer
Schicht 1	Bitübertragungsschicht	Physical Layer

Tabelle 9.2: Das OSI–Schichtenmodell

9.3 Klassifizierung von Rechnernetzen

Wichtig für die Klassifizierung von Rechnernetzen sind Aspekte der Netz–Topologie und des Netzzugangs. Im Folgenden sollen Netze bezüglich dreier Merkmale klassifiziert werden.

9.3.1 Nach der Topologie

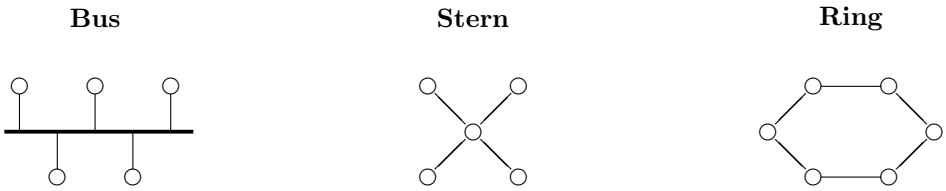


Abbildung 9.1: Wichtige LAN–Topologien

Punkt–zu–Punkt–Netzwerke: Das Rechnernetz besteht aus einer Anzahl von Verbindungen zwischen je zwei Rechnern. Verschiedene Topologien sind möglich: Stern, Ring, Baum, Bus, Vollständig vermascht, (siehe auch Abb. 9.1).

Broadcast–Netzwerke: Alle Rechner im Netz teilen sich einen gemeinsamen Kommunikationskanal. Daten, die von einem Rechner abgeschickt werden, können potentiell von allen anderen im Netz befindlichen Rechnern empfangen werden (z.B. Satellitenfunk). Durch Einfügen der Empfängeradresse wird die Nachricht eindeutig zugeordnet. Lokale Netze werden vorwiegend als Broadcast–Netzwerke betrieben.

9.3.2 Nach der Vermittlungstechnik

Leitungsvermittelte Netze: In leitungsvermittelten Netzen wird vom Sender ausgehend eine Verbindung zum Empfänger durchgeschaltet, die für die Dauer der Übertragung aufrechterhalten wird. Die Unterstützung der Schichten 1 bis 3 (Tabelle 9.2, siehe auch Abschnitt 9.4 auf Seite 144) hinsichtlich Multiplexen, Fehlererkennung und Flusskontrolle ist hier nicht gegeben, weshalb sich beide Kommunikationspartner gleich verhalten müssen. Ein Beispiel hierfür ist der Übertragungsdienst der Post DATEX–L.

Die Übertragungskosten werden nach der Dauer der Verbindung berechnet. Leitungsvermittelte Netze sind daher nützlich für die Übertragung großer Datenmengen. Sie bestehen aus eher historischen Gründen für ältere Anwendungen.

Paketvermittelte Netze: In paketvermittelten Netzen werden identifizierbare Datenpakete durch das Netz geschickt, dabei unter Umständen auch zwischengepuffert (Store–and–Forward Netz). Ein Anwendungsbeispiel ist der DATEX–P–Dienst der Post. Im wesentlichen unterscheiden wir zwei Arten der Paketvermittlung:

- verbindungsorientiert: Es wird explizit eine virtuelle Verbindung zum Zielrechner aufgebaut.
- verbindungslos: Jedes Datenpaket wird einzeln transportiert. Da Pakete auf unterschiedlichen Wegen transportiert werden können, muss hier auf die Reihenfolgeerhaltung geachtet werden.

Die Kosten berechnen sich nach der Anzahl der verschickten Pakete (siehe auch Transportschicht, Abschnitt 9.4.4, Seite 144).

9.3.3 Nach der Reichweite

Da die Einteilung nach der Reichweite üblich ist, werden wir genauer darauf eingehen.

9.3.3.1 Lokale Netze — Local Area Networks (LAN)

Lokale Netze lassen sich nicht eindeutig definieren. Wir versuchen es hier wieder mit einer Definition von Tanenbaum [15]:

- Ein lokales Netz untersteht einer einzigen Organisation bzw. einem einzigen Unternehmen.
- Es erstreckt sich über ein Gebiet von wenigen Quadratkilometern.
- Es hat eine hohe Übertragungsrate von 1–200 Mbit/sec.

Gerade weil sich LANs im Besitz einer einzigen Organisation befinden, spielen Kompatibilitätsprobleme keine große Rolle, sind Standards von geringerer Bedeutung. Hier eine Übersicht über gängige Konfigurationen von LANs:

- **Private Automated Branch Exchange (PABx):** Bei einer PABx handelt es sich um ein Netz in Sterntopologie mit einer Untervermittlungsanlage im Mittelpunkt. PABx arbeiten leitungsvermittelt.
- **Ring:** Jeder Rechner im Ring besitzt eine Sende- und eine Empfangseinrichtung. Daten kreisen im Ring, bis sie der Empfänger oder Sender aus dem Ring herausnimmt. Ein entscheidendes Problem bei Ringen ist die Frage des Netzzugangs, d.h. zu welchem Zeitpunkt ein Rechner Daten senden darf, ohne dass es zu einer Datenkollision kommt. Verschiedene Protokolle sind bekannt:
  - **Token–Ring:** Ein ausgezeichnetes Bitmuster (Token) kreist solange im Ring, bis ein Rechner mit Sendewunsch es entfernt. Nach Senden seiner Daten plaziert er den Token in den Ring zurück.
  - **Slotted–Ring:** Im Ring kreisen freie und belegte Slots (Container). Hat ein Rechner einen Sendewunsch, belegt er den nächsten freien Slot.
  - **Register–Insertion–Ring:** Ein Rechner mit Sendewunsch reiht alle ankommenden Daten in ein Schieberegister. Er kann dann solange Daten senden, bis das Schieberegister voll ist. Dann muss der Inhalt des Registers wieder auf den Ring geschickt werden.
- **Bus:** Alle Rechner teilen sich eine gemeinsame Leitung. Wie bei den Ringkonfigurationen muss eine Datenkollision aufgefangen werden. Bekannte Protokolle sind:
  - **Aloha:** Jeder Rechner sendet, wann er will. Dazu enthält jedes Datenpaket eine Prüfsumme (CRC), so dass der Empfänger eine Kollision feststellen kann. Er verweigert dann die Bestätigung des Pakets, und der Sender wiederholt die Übertragung nach einer vereinbarten Wartezeit. Dieses Verfahren wurde zum ersten Mal in Hawaii erprobt.
  - **CSMA:** Carrier Sense Multiple Access. Jeder Rechner hört die Leitung ab. Falls keine Daten auf der Leitung sind, kann gesendet werden. Eine Kollision kann nur in den ersten Bits eines Pakets auftreten.
  - **CSMA/CD:** Carrier Sense Multiple Access/Collision Detection. Auch während des Sendevorgangs hört der Sender ab. Dadurch wird eine Kollision direkt beim Sender festgestellt. Das Verfahren ist effizienter als CSMA, es wird beim IEEE Standard 802.3 (auch Ethernet genannt) verwendet.

Eine Einordnung der Protokolle für LANs in das im Folgenden beschriebene OSI–Modell ist schwierig, da sie unabhängig von den ISO–Standards entstanden, und Standards wegen der Nutzung als Inhouse–Anwendungen nicht so wichtig waren. Da die Kommunikationssoftware weniger komplex ist, war eine Einteilung in Schichten nicht notwendig.

9.3.3.2 Fernnetze — Wide Area Networks (WAN)

Da WANs häufig ihre Konfiguration ändern, haben sie selten regelmäßige, sondern eher allgemeine Topologien. In diesem Bereich sind internationale Standards wie das OSI–Referenzmodell besonders wichtig.

Um einerseits unterschiedliche Netze miteinander verbinden, andererseits inhomogene Netze betreiben zu können, werden einzelne Netzteile über **Gateways** gekoppelt. Gateways setzen verschiedene Protokolle, verschiedene Kodierung, verschiedene elektrische Eigenschaften, etc. von Netzen ineinander um. In der Praxis werden auch Zwischenformate wie TCP/IP, die im LAN–Bereich entstanden sind, verwendet. Diese beiden Protokolle sind in den Schichten 3 und 4 anzusiedeln, wurden aber vor den ISO–Standards entwickelt.

Ist ein WAN als Punkt–zu–Punkt–Netzwerk ausgelegt, ist die Wegewahl– und Flusskontroll–Problematik entscheidend. Bei Broadcast–Netzen sind Empfängererkennung und Sendekontrolle zu realisieren (Siehe auch die Ausführungen zu LANs).

9.4 Das OSI–Schichtenmodell

Das OSI–Schichtenmodell, das wir im Folgenden kurz skizzieren, ist im Standard ISO–7498, der sogenannten Basisnorm, beschrieben. Das OSI–Modell besteht aus 7 streng hierarchisch angeordneten Schichten, die in Abb. 9.2 dargestellt sind (siehe auch Tabelle 9.2, Seite 140)

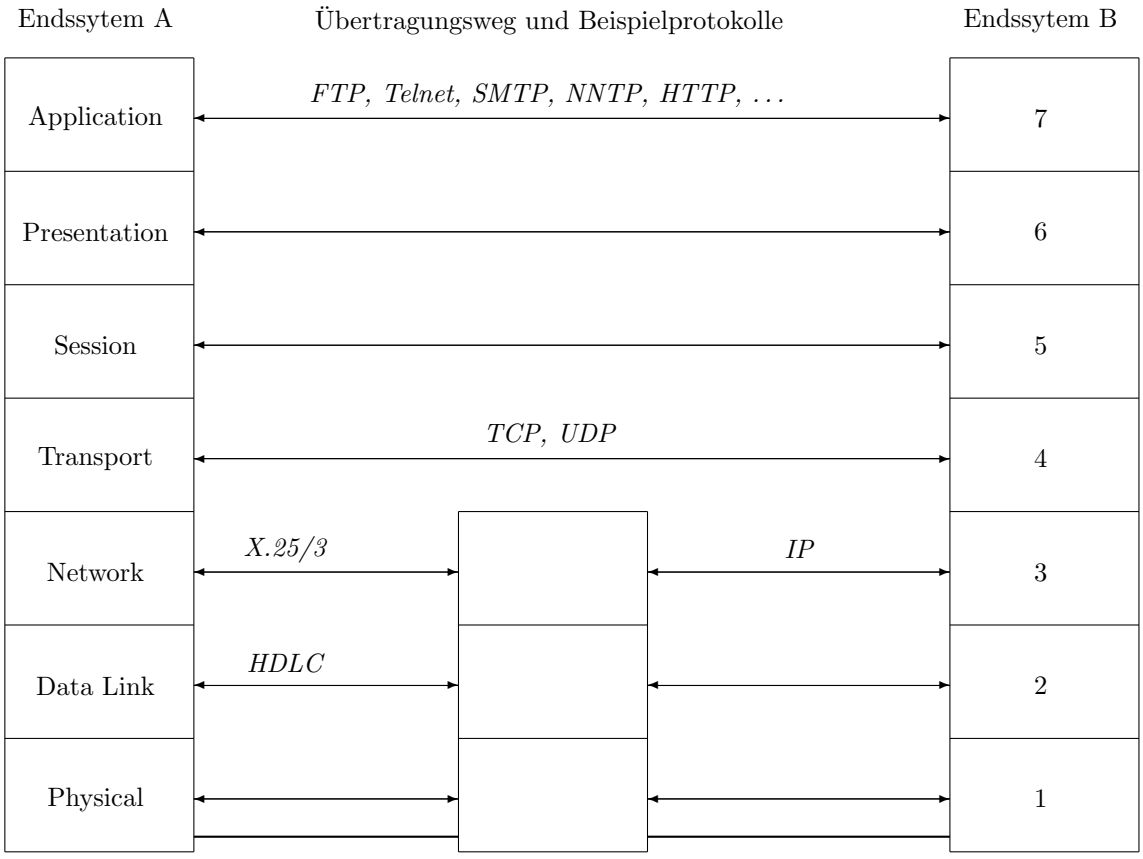


Abbildung 9.2: Das OSI–Schichtenmodell

Jede Schicht erbringt bestimmte Leistungen und bietet diese der übergeordneten Schicht als sogenannte

Dienste an. Eine Schicht erbringt ihren Dienst, indem sie den Dienst der direkt untergeordneten Schicht in Anspruch nimmt. Die Bitübertragungsschicht kann natürlich keine untergeordnete Schicht benutzen; sie setzt direkt auf dem Übertragungsmedium auf. Wir wollen uns jetzt die Aufgaben der einzelnen Schichten, die in Bild 9.2 und Tabelle 9.2 dargestellt sind, etwas genauer ansehen.

9.4.1 Schicht 1 — Bitübertragungsschicht

Die Bitübertragungsschicht ist für die Übertragung einzelner Bits zwischen Kommunikationssystemen über ein konkret vorhandenes Medium zuständig. In Tabelle 9.3 sind Übertragungsmedien aufgelistet, die in Rechner–Netzwerken Verwendung finden. Die Übertragung ist nicht notwendigerweise transparent. Unter anderem müssen Überlegungen angestellt werden hinsichtlich Anzahl und Bedeutung der Leitungen, Signalspannungen, simplex/halbduplex/duplex Übertragung, analoge/digitale Übertragung, parallele/serielle Übertragung, synchrone/asynchrone Übertragung, etc. Die bestehenden Standards lassen sich in Standards für analoge (V–Serie, z.B. V.24) und digitale (X–Serie, z.B. X.21) Netze einteilen.

Rundfunk	Bodenstationen tauschen über ein festgelegtes Frequenzband Datenpakete aus.
Satellitenfunk	Analog zum Rundfunk; allerdings wird hier ein geostationärer Satellit dazwischen geschaltet, geeignet für den Fernbereich. Im Vergleich zum Rundfunk entstehen durch die stärkeren Sendestationen und den Satelliten höhere Kosten.
Telefonkabel	Analoge oder digitale Datenübertragung. Zur Kommunikation wird das bestehende und sehr gut ausgebaute Telefonnetz verwendet (Datex-P, Datex-L der Post, ISDN).
Koaxialkabel	Digitale oder analoge Datenübertragungsarten. Anwendung in lokalen Netzwerken bis 1 km wie Ethernet mit Übertragungsleistungen um die 10 $MBit/s$ und höher.
Glasfaser	Lichtpulse dienen zur Datenübertragung. Besonders störungsfreie schnelle Datenübertragung, aber sehr hoher Preis.

Tabelle 9.3: Übertragungsmedien von Rechnernetzwerken

9.4.2 Schicht 2 — Abschnittssicherungsschicht

Die Abschnittssicherungsschicht gewährleistet eine transparente, von Übertragungsfehlern freie Verbindung zwischen direkt verbundenen Rechnern. Hier werden Fehlererkennung/Fehlerbehebung, Flusskontrolle, Zugriffsverfahren auf das Übertragungsmedium, Topologie usw. festgelegt.

9.4.3 Schicht 3 — Vermittlungsschicht

Die Vermittlungsschicht dient der Übertragung von Informationen zum Zielrechner. Im Unterschied zu den unteren Schichten müssen Rechner nicht direkt gekoppelt sein. Eine Schlüsselfunktion ist die Wegewahl (Routing) von Datenpaketen zum Zielrechner. Desweiteren protokolliert diese Schicht mit, wieviele Datenpakete, Zeichen oder Bits von jedem Teilnehmer gesendet werden.

9.4.4 Schicht 4 — Transportschicht

Die Transportschicht liefert eine verbindungsorientierte oder verbindungslose, kostenoptimale Datenübertragung, in der der Benutzer die Qualität des Dienstes wählen kann. Ihre Basisfunktion ist, Daten aus

der Kommunikationssteuerungsschicht zu übernehmen, diese in kleine Teile zu zerlegen, der Vermittlungsschicht zu übergeben und dafür zu sorgen, dass diese Teile am anderen Ende wieder korrekt zusammengesetzt werden. Diese Schicht muss dafür sorgen, dass das effizient und hardware–unabhängig erfolgt. Es werden Verfahren für Multiplexing, Resequencing und verschiedene Arten der Kostenoptimierung und zur Anhebung der Dienstqualität angewandt.

9.4.5 Schicht 5 — Kommunikationssteuerungsschicht

Diese Schicht erlaubt Benutzern Verbindungen (Sessions) auf verschiedenen Rechnern aufzubauen. Beispiele sind Login auf anderen Rechnern oder die Möglichkeit zum Filetransfer zwischen zwei Rechnern. Desweiteren wird die Synchronisation von Kommunikationsbeziehungen, z.B. für Filetransfer von einigen Stunden über viele Rechner hinweg, sowie Dialogmöglichkeiten mit dem Benutzer, ermöglicht.

9.4.6 Schicht 6 — Darstellungsschicht

Die Darstellungsschicht befasst sich nicht mit konkreten Übertragungsproblemen, sondern mit der Syntax (formaler Aufbau) und Semantik (Bedeutung) der zu übertragenden Information. Ein Beispiel ist die Umwandlung der Information in einen Standard, der zwischen den Kommunikationspartnern gewählt wurde, wie Formate für Datentypen (integer, real, etc.) und Datenstrukturen.

9.4.7 Schicht 7 — Anwendungsschicht

Sie macht Anwendungsprozessen die Dienste des OSI–Referenzmodells zugänglich und stellt verschiedene häufig benötigte Dienste zur Verfügung. Darunter fallen alle Anwendungsprogramme wie ftp, telnet, rlogin, rsh, rcp, mail, nis und verschiedene Netzwerk–Filesysteme wie NFS. Jedes Anwendungsprogramm, das im OSI–Modell Schichten verwendet, ist der Anwendungsschicht zugeordnet. Dies ist unabhängig davon, welche Dienste aus welcher Schicht verwendet wurden, d.h. nicht alle Schichten des OSI–Modells müssen für eine Anwendung verwendet werden.

Die Datenpakete, die zwischen zwei gleichrangigen Schichten ausgetauscht werden, heißen Protocol Data Units (PDU). Die zusätzlichen Informationen, die zwischen zwei benachbarten Schichten ausgetauscht werden, um ihren Dienst zu erfüllen, heißen Protocol Control Informations (PCI).

Wie läuft nun nach dem OSI–Modell ein bestimmter Kommunikationsvorgang ab? Je zwei Schichten A_n und B_n kommunizieren miteinander unter Einhaltung bestimmter Regeln und Konventionen, bekannt als das Schicht–n–Protokoll. Wenn eine Schicht A_n einer Schicht B_n Informationen übermitteln will, übergibt sie diese Informationen der Schicht A_{n-1}. Die Schicht A_{n-1} fügt zu diesen Informationen ihre eigene PCI hinzu, die sie zur Abwicklung ihres Schicht–(n-1)–Protokolls benötigt, hinzu. Anschließend übergibt sie die Gesamtinformationen an die Schicht A_{n-2}, usw. bis auf Schicht 1 der physikalische Übertragungsvorgang stattfindet. An der anderen Seite angekommen, werden die Kontrollinformationen auf jeder Schicht wieder entfernt, bis auf Schicht B_n die eigentlichen Informationen vorhanden sind.

9.5 Das Internet–Schichtenmodell

Neben dem OSI–Referenzmodell gibt es eine weitere Sicht auf auf die unterschiedlichen, an einer Netzwerkverbindung beteiligten Schichten: das Internet–Schichtenmodell. Dieses Modell fasst die sieben Schichten des OSI–Schichtenmodells zu insgesamt vier Schichten zusammen. Der Grund hierfür ist eine vereinfachte, an die Gegebenheiten einer Internetapplikation angepassten Sicht auf die Schichten.

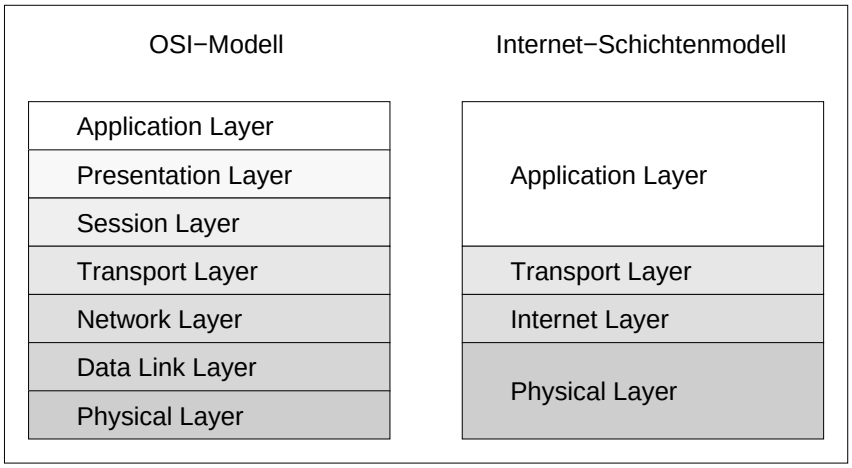


Abbildung 9.3: Das OSI–Referenzmodell im Vergleich zum Internet–Schichtenmodell

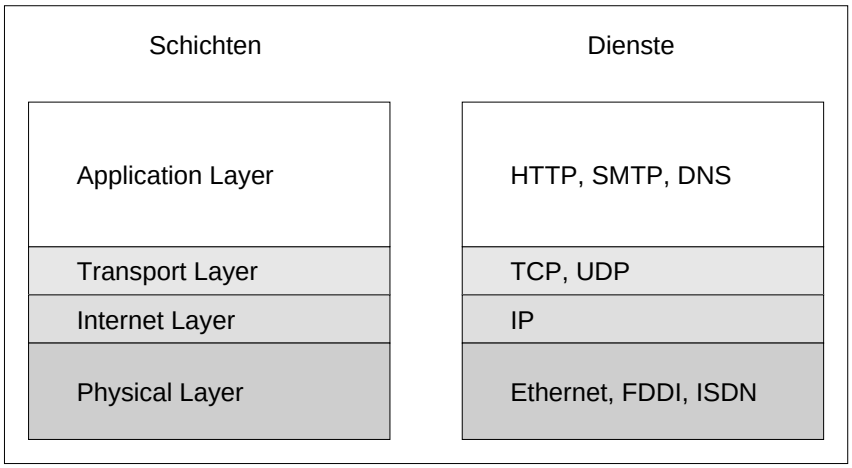


Abbildung 9.4: Beispielhafte Dienste des Internet–Schichtenmodells

Konkret werden die OSI–Schichten 1 und 2 als *physische Schicht*, sowie die OSI–Schichten 5 bis 7 als *Applikationsschicht* zusammengefasst.

Eine Gegenüberstellung der Schichten des OSI–Referenzmodells und des Internet–Schichtenmodells liefert Abbildung 9.3.

Einige Beispiele für Dienste der Internet–Schichten zeigt Abbildung 9.4.

9.6 Netzwerkstandards

In diesem Abschnitt wollen wir etwas über konkrete, marktgängige Netzwerkstandards erfahren. Es sind dies insbesondere *Ethernet*, *Token Ring* und *TCP/IP*.

9.6.1 Die IEEE Standards für lokale Netze

Die Vereinigung IEEE (*Institute of Electrical and Electronics Engineers, Inc.*) hat einige Standards für lokale Netze entwickelt, die unter dem Namen IEEE 802 bekannt sind. Diese Standards unterscheiden sich im Physical Layer, sind aber kompatibel im Data Link Layer. Die IEEE–Standards sind von allen wichtigen Standardisierungsgremien übernommen worden. Es wurden folgende Standards erstellt:

- IEEE 802.3 und Ethernet: Dieser Standard beschreibt ein LAN mit CSMA/CD. Die Grundlage dieses Standards war eine Entwicklung von Xerox, das Ethernet. Der Unterschied von Ethernet zu 802.3 ist, dass hier eine ganze Familie von CSMA/CD–Systemen mit Geschwindigkeiten von 1 bis 10 MBit/s beschrieben wird. Zur Erkennung von Kollisionen durch gleichzeitiges Senden von Teilnehmern wird ein analoges Verfahren zur Fehlererkennung benutzt. Ethernet ist ein spezifisches Produkt nach diesem Standard, wird aber oft mit diesem verwechselt. Der Name Ethernet bezieht sich mehr auf das Übertragungsmedium, das Kabel.

  Es werden zwei Arten von Koaxialkabel verwendet, das Thick– und das Thin–Ethernet. Das dünne Ethernet ist, wie der Name sagt, im Kabel dünner und verwendet BNC–Stecker als Anschlüsse. Es kann allerdings nur für kürzere Entfernungen verwendet werden. Das dickere Ethernet verwendet einen Transceiver mit einem Dorn, der alle 2,5 m in das Kabel gesteckt werden kann.

- IEEE 802.4 Bus: Während 802.3 vorwiegend in Büros Anwendung fand, waren große Firmen wie General Motors an Fabrikautomation interessiert, also an höhere Sicherheit und Echtzeitfähigkeiten. So wurde 802.4 entwickelt. Physikalisch gesehen ist es ein lineares Kabel, an dem die Stationen angeschlossen sind. Logisch gesehen handelt es sich um einen Ring, in dem jede Station die Adresse seines Nachbarn kennt. Ein Token wird jetzt direkt von einem Nachbarn zum nächsten gesandt. Nur der Inhaber eines Tokens ist in der Lage, Nachrichten auf dem Bus zu verschicken. Dadurch werden Kollisionen, wie unter 802.3 vermieden. Dieser Standard ist wesentlich komplexer als 802.3. Die Geschwindigkeiten des Token Bus liegen bei 1,5 und 10 MBit/s. 802.4 verwendet als Übertragungsmedium ein 75 Ω Breitbandkoaxialkabel.

- IEEE 802.5 Token Ring: Der Hersteller IBM hatte sich schon sehr früh für einen Ring als LAN entschieden. Der Standard 802.5 ist kompatibel mit dem IBM Token Ring. Ein Ring ist eine Sammlung von Ringschnittstellen, die jeweils eine Punkt zu Punkt Verbindung enthalten. Im Ring kreist ein spezielles Bitmuster, der Token. Wenn eine Station Daten senden will, nimmt sie sich den Token und sendet Daten. Nur eine Station kann den Token vom Ring entfernen. Nach dem Senden muss der Token von der Station wieder erzeugt werden, der dann wieder im Ring kreist. Dadurch kann der Ring selbst unter höchster Last eine 100 prozentige Effizienz erreichen. 802.5 benutzt als Übertragungsmedium ein verdrilltes Kupferkabel und erreicht Übertragungsgeschwindigkeiten von 1 bis 16 MBit/s.

9.6.2 Beispiel einer Anwendung der Schicht 3: IP

Eine Anwendung der Vermittlungsschicht ist das Internet Protocol (IP). Dieses Protokoll wurde Anfang der achtziger Jahre eingeführt und hat sich im Zusammenhang mit dem TCP–Protokoll (siehe unten) im Netzwerkbereich sehr stark durchgesetzt.

Das IP–Protokoll ist ein verbindungsloses Protokoll, das auf der Idee von transparenten, aber nicht notwendigerweise zuverlässigen Internet Datagrammen basiert, die vom Senderechner zum Empfangsrechner über beliebige Netzwerke hinweg transportiert werden.

Die Transportschicht empfängt Nachrichten und teilt sie in Datagramme der Größe 64 kBytes auf. Jedes Datagramm wird über das Internet übertragen, wobei ein Datagramm in kleinere Einzelteile aufgeteilt werden kann. Wenn alle Teile am Ziel angekommen sind, werden sie von der Transportschicht wieder zur Originalnachricht zusammengebaut.

9.6.3 Beispiel einer Anwendung der Schicht 4: TCP

Wir wollen die heutzutage wichtigste Implementierung der Transportschicht, das Transmission Control Protocol vorstellen. TCP wird meist im Zusammenhang mit dem Protokoll IP der Vermittlungsschicht genannt.

Die Transport–Entity TCP übernimmt beliebig lange Nachrichten von Benutzerprozessen und teilt diese in Größen von maximal 64 kBytes auf. Die einzelnen Teile werden als Datagramme verschickt. Die Vermittlungsschicht gibt keine Garantie, dass die Daten korrekt übertragen werden, so dass TCP dafür sorgt, den Datagrammen ein Zeitlimit vorzugeben und sie notfalls wiederholt sendet. Da die Datagramme nicht notwendigerweise in der korrekten Reihenfolge ankommen, muss TCP sie wieder korrekt zusammensetzen.

TCP ist ein Nicht–OSI–Produkt und hat, wie die meisten dieser Kategorie, eine gut definierte Benutzerfläche. Es gibt Aufrufe zum Aufbau und Abbau von Verbindungen (connect, listen), zum Senden und Empfangen von Daten und zur Statusabfrage der Verbindung.

Vergleich von OSI Schicht 4 und TCP

Ein Vergleich zwischen dem OSI–Schichtenmodell und TCP ist notwendig, da beide Gemeinsamkeiten aber auch Unterschiede haben. Gemeinsamkeiten sind: Beide Protokolle bieten eine zuverlässige, verbindungsorientierte, end–to–end Transportleistung auf einem nicht zuverlässigen Netzwerk, das Datenpakete verlieren, zerstören, speichern und duplizieren kann. Beide haben eine Phase zum Herstellen einer Verbindung, zur Übertragung der Daten und zur Auflösung der Verbindung.

Es gibt jedoch auch wesentliche Unterschiede zwischen beiden. Die OSI Schicht 4 benutzt neun Datenpaket–Typen (TPDU), TCP nur eine. TCP ist dadurch zunächst einfacher, benötigt aber vielmehr Platz für seinen Kopf, da dort alles, was in den OSI–TPDU's enthalten ist, auch dort enthalten sein muss. Differenzen bestehen auch bei der Behandlung einer Kollision, die durch die gleichzeitige Anforderung von Diensten durch zwei Prozesse entsteht, sowie dem Adressformat. Es gibt noch weitere Unterschiede zwischen TCP und OSI 4, die hier aber nicht mehr genannt werden sollen.

9.7 Zusammenfassung

Ein Computer–Netzwerk ist eine Ansammlung zusammenhängender, selbständiger Computer. Vorteile von Netzwerken sind effektive Nutzung von Hardware– und Software–Ressourcen, die zentral gewartet werden können, Zuverlässigkeit und günstiges Preis–Leistungsverhältnis. Netzwerke finden neben der Versorgung von Workstations Anwendung z.B. bei der Dateiübertragung, bei remote login und bei electronic mail.

Die zum Betrieb eines Netzwerkes erforderliche Netzwerk–Software hat die Aufgaben Verbindungsaufbau und -abbau, Wegewahl, transparente Übertragung, Flusskontrolle, Multiplexing und Splitting/Recombining sowie Übertragungssicherung.

Gesichtspunkte zur Klassifizierung von Rechnernetzen sind Topologie (Stern, Ring, Bus), Übertragungsmedium (einfache Kabel, Koaxialkabel, Lichtwellenleiter), Zugriffsverfahren (Token Access, CSMA/CD). Durchgesetzt haben sich der Token Ring (Ringsysteme mit Token Access) und Ethernet (Bussystem mit CSMA/CD), beide auf der Basis von Koaxialkabeln.

Ein allgemeiner Standard ist das OSI–Referenzmodell für Software für die Kommunikation in offenen Systemen. Es beschreibt universell eine logische Struktur aus sieben hierarchischen Schichten für die Aufgaben der Datenkommunikation. Jede Schicht erbringt einen definierten Dienst, wobei sich die unterste Schicht mit der eigentlichen Datenübertragung befasst.

Im Bereich der Unix–Netze sind TCP/IP weit verbreitete Kommunikationsprotokolle, die den OSI–

Schichten 3 (Vermittlung) und 4 (Transport) zugeordnet werden können.[1]

9.8 Fragen und Übungen zu Kapitel 9

F 9.1 Was ist ein Computer—Netzwerk?

F 9.2 Welche Vorteile hat ein Computer–Netzwerk?

F 9.3 Was ist das OSI–Referenzmodell?

F 9.4 Ordne die Begriffe Ethernet, Token–Ring, TCP, IP.

F 9.5 Welche Aufgaben hat eine Kommunikationssoftware?

9.9 Antworten zu den Fragen und Übungen zu Kapitel 9

A 9.1 Ein Computer–Netzwerk ist eine Ansammlung zusammenhängender, selbständiger Computer.

A 9.2 Rechnernetze bieten:

- Bessere Ausnutzung von Resourcen: Plattenspeicher, Peripherie, Rechenleistung usw.
- Zugriff auf gemeinsame Daten
- Höhere Zuverlässigkeit
- Günstiges Preis/Leistungsverhältnis

A 9.3 Das OSI–Referenzmodell ist ein Referenzmodell für Software für die Kommunikation in offenen Systemen. Es beschreibt universell eine logische Struktur aus sieben hierarchischen Schichten für die Aufgaben der Datenkommunikation.

- Application Layer — Anwendungsschicht
- Presentation Layer — Darstellungsschicht
- Session Layer — Kommunikationssteuerungsschicht
- Transport Layer — Transportschicht
- Network Layer — Vermittlungsschicht
- Data Link Layer — Abschnittssicherungsschicht
- Physical Layer — Bitübertragungsschicht

A 9.4 Ethernet beschreibt zuerst einmal Hardware, nämlich ein Koaxialkabel. Man unterscheidet nach Thin und Thick Ethernet. Zum Ethernet gehört auch ein Protokollstandard: CSMA/CD (Mehrfachzugriff mit Trägerabtastung und Kollisionserkennung), ein Bus–Protokoll.

TCP und IP sind Kommunikationsprotokolle, die meistens zusammen genannt werden. Es gibt Gemeinsamkeiten mit den OSI–Schichten $3 + 4$ (Vermittlung und Transport). TCP/IP hat sich als weitverbreiteter Standard etabliert.

Der Token Ring ist ein anderer Netzwerkstandard, wobei das Netz als Ring ausgeführt ist. Ein spezielles Bitmuster, das Token, kreist im Ring. Nur wer das Token hat, darf senden.

A 9.5 Kommunikationssoftware kümmert sich um Wegewahl, transparente Übertragung, Flusskontrolle, Multiplexing und Splitting/Recombining sowie um Übertragungssicherung in Netzwerken.

[1] Dieses Kapitel basiert auf dem entsprechenden Kapitel des Skripts v. Prof. Biedl, FB 13.
W.W. Baumann, Januar 1991, überarbeitet und aktualisiert von Chr. Böhning, Februar 1998

Kapitel 10

Grafische Datenverarbeitung

10.1 Allgemeine Einführung

Die Programme, die wir bisher geschrieben haben, haben ihre Ergebnisse in mehr oder weniger umfangreichen Zahlenkolonnen, hin und wieder bereichert um einige Kommentarstrings, bereitgestellt. Die Sichtung und Interpretation dieser Zahlenkolonnen kann oft sehr mühsam sein, man wünscht sich eine andere Art der Präsentation. Dieser und andere Aspekte der Mensch–Maschine–Kommunikation lassen sich wesentlich dadurch verbessern, daß der Rechner seine Ergebnisse in Diagrammen und Bildern bereitstellt, weil bildhafte Darstellungen für den Menschen eine anschauliche Arbeitsgrundlage bilden.

Die Grafische Datenverarbeitung (GDV) befaßt sich als Teilgebiet der Angewandten Informatik mit der rechnerunterstützten Generierung, Speicherung, Darstellung und Manipulation von Bildern. Da Rechner mit Zahlen arbeiten und als Ergebnis Zahlen liefern, müssen zum einen alle grafischen Informationen für die rechnerinterne Verarbeitung in Zahlen umgewandelt werden, zum anderen müssen die Ergebnisse in grafische Informationen umgewandelt werden. Unter grafischer Information werden alle Arten von bildlicher Information verstanden, die aus Kurven oder Rasterbildern bestehen.

In der grafischen Datenverarbeitung unterscheiden wir die passive und die interaktive Arbeitsweise. Die **passive grafische Datenverarbeitung** umfaßt die Ausgabe grafischer Daten. Der Betrachter der erstellten Bilder hat dabei keinen direkten Einfluß auf die Bildgenerierung. Bei **interaktiver GDV** steuert der Mensch die Kommunikation mit dem Rechner durch Kommandos. Er kann sowohl eine alphanumerische als auch eine grafische Ausgabe erhalten. Interaktives Arbeiten haben wir bereits im Zusammenhang mit der Arbeit an PCs kennengelernt. Die Entwicklung von CAD–(engl. **c***omputer* **a***ided* **d***esign*)–Systemen hat die grafische Datenverarbeitung stark beeinflußt. So können z.B. Konstruktionszeichnungen relativ einfach und schnell mit Hilfe von Rechnern und automatischen Zeichenmaschinen erstellt werden. Daher wird die GDV in **vielen Gebieten** als Hilfsmittel herangezogen:

- **CAE** (engl. **c***omputer* **a***ided* **e***ngineering*): dieses beinhaltet CAD, dessen unterste Stufe computergesteuertes Zeichnen ist. Beim computergestützten Entwurf und Konstruieren werden 2D– und 3D–Objekte (z.B. Flächen, Körper etc.) erzeugt, gespeichert und manipuliert. Es kann sich dabei um Drahtmodelle (engl. *wire frames*) handeln, ohne Entfernen von verdeckten Kanten (engl. *hidden lines*), oder um Festkörpermodellierung, mit Entfernen von verdeckten Kanten. Beim CAM (engl. **c***omputer* **a***ided* **m***anufacturing* — computergestützte Fertigung) gehen CAD–Ergebnisse in einen teilautomatischen Entwurf ein.
- **Visualisierung**: Sichtbarmachung von Prozessen und Systemen (mechanisch, elektrisch, chemisch), die vorher der eigenen Vorstellungskraft überlassen waren, durch Simulation: Vorhersage von Systemverhalten durch Beschreibung mit mathematischen Formeln und Programmierung im Computer. Auch dynamisches Verhalten kann gezeigt werden, z.B. Umströmung eines Flugzeugs.

- **Architektur und Bauingenieurwesen**: Modellierung von Landschaften, Gebäuden, Brücken, Straßen. Ansichten von und Einsichten in neue Gebäude, z.B. Fabriken, und deren Einpassung in die Umgebung (Stadtplanung); Kartographie.
- **Entwurf elektrischer und elektronischer Systeme**: Entwurf von Platinen für Schaltungen und Mikroprozessorchips: Anzeige der Leiterbahnen in den verschiedenen Schichten; Simulation des Verhaltens des Chips.
- **Automatisierung und Robotersysteme**: Automatische Fabriken, Robotermontage und Produktionssysteme: Planung der Roboterbewegung und Produktionssysteme: Planung der Roboterbewegung, grafische Kollisionskontrolle in der Roboterzelle.
- **Produktionsentwurf, Grafische Entwürfe und Kunst**
- **Unterhaltung und Werbung**: Animation.
- **Ausbildung und Schulung**: z.B. Flug- oder Fahrsimulator.
- **Präsentationsgrafik**
- **Steuerung und Regelung**: Schematische Anzeige eines Prozesses und seines gegenwärtigen Zustandes, z.B. Temperaturen und Drücke in Rohrleitungen und Kesseln.
- **Wissenschaft und Forschung**: z.B. Darstellung von Dingen, die man nicht sehen sondern sich nur vorstellen kann: Moleküle.
- **Medizinische Systeme**: Darstellung des Körpers und seiner Organe zu Diagnose- und Operationszwecken, z.B. mit Hilfe der Computertomographie.

10.2 Die grafische Darstellung

Die **Erzeugung einer computergrafischen Darstellung** beginnt mit einem Modell der darzustellenden Szene. Das Modell wird dann zusammengesetzt aus sogenannten Primitiven (das sind die Bausteine für ein Bild) — z.B. Linien, gefüllte Flächen, Marker — und ihren Attributen — Liniendicke, Farbe usw. Die Bildelemente können auf verschiedene Art dargestellt werden. Im einfachsten Fall kann das **monochrom** sein, z.B. schwarz/weiß, erzeugbar z.B. mit schwarzem Stift auf weißem Papier. Abstufungen können durch Schattieren (z.B. schwarze Punkte auf weißem Grund oder verschiedene Helligkeiten (Graustufen) erzeugt werden. Das Attribut Farbe beinhaltet noch eine weitere Dimension der Information, die das Bild liefert. Farbe kann als Erweiterung der Graustufen betrachtet werden. Man unterscheidet zwei Darstellungs- und Verarbeitungsmöglichkeiten: **linienorientierte** Darstellung und **flächenhafte** Farb- und Grautondarstellung, d.h. Bilder setzen sich aus Linien oder Punkten (entartete Linien) zusammen. Diese haben eine Lage, eine Länge und eine Richtung: **Vektoren**. Bildpunkte (**pixel**) als Bildelemente sind die kleinsten adressierbaren Einheiten auf dem Bildschirm. Sie können Schattierungen erzeugen durch ihre Abwesenheit oder ihre Helligkeit. Aus Pixeln können auch Linien zusammengesetzt werden.

10.3 Das Grafiksystem

Ein grafisches Datenverarbeitungssystem, kurz **Grafiksystem** setzt sich aus Hardware- und Softwarekomponenten zusammen. Da ist zum einen der Rechner, auf dem das Anwenderprogramm läuft. Das Anwenderprogramm beinhaltet die Algorithmen und die Logik, die zur Erfüllung einer bestimmten Aufgabe durch das Grafiksystem notwendig sind. Es liefert auch eine Schnittstelle zum Anwender in der Form von Anweisungen, Antworten und Echos (Eingabe–Wiedergaben), und es akzeptiert und verfiziert alle Anwender–Eingaben. Die Anwendungs–Software kommuniziert direkt mit dem Grafik–Paket mit

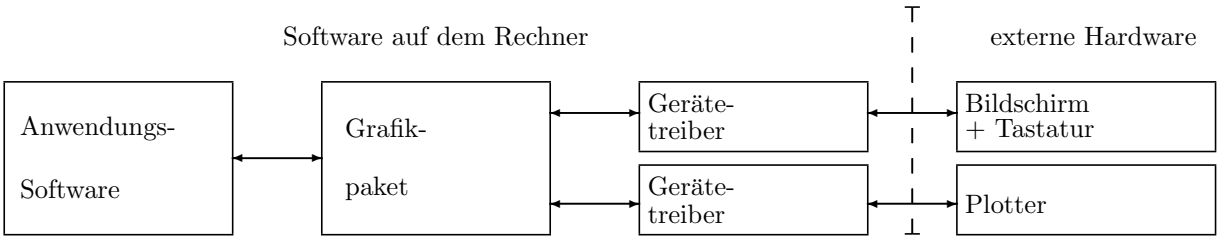


Abbildung 10.1: Struktur eines typischen Grafiksystems

Hilfe von logischen Eingabe- und Anzeigenfunktionen. Sie sind definiert durch das virtuelle Grafikgerät, welches die verwendete Grafikbibliothek benutzt.

Das Grafiksystem kann auch zusätzlich Spezialsoftware für nicht–grafische Aufgaben enthalten. Das Grafikbibliothekspaket interpretiert und verwaltet die Grafikfunktionen, die die Anwendung erfordert und verwendet die Gerätetreiber, um die logischen Grafikfunktionen mit den realen Geräten auszuführen. Der Gerätetreiber wiederum setzt diese Anweisungen um und kommuniziert direkt mit der Hardware über das Schnittstellenprotokoll dieser Hardware. Dabei deutet sich auch schon das Hauptproblem der Computergrafik an: **Abhängigkeit von der Hardware**.

10.4 Standards der Computergrafik

Das Problem der Hardwareabhängigkeit muß durch **Standardisierung** gelöst werden. Die für die GDV entwickelte Hardware wird in so großer Vielfalt angeboten, daß es abgesehen von schlüsselfertigen Systemen üblich ist, Rechner und verschiedene Peripheriegeräte unterschiedlicher Fabrikate zu verwenden. Die gesamte Software, insbesondere die Grafiksoftware, muß von der Hardware unabhängig sein, so daß sie auf den verschiedenen Rechnern und Geräten einsetzbar ist. Die **Portabilität** der Software stützt sich auf die Rechner- und Geräteunabhängigkeit. Um für eine immer wiederkehrende Menge von grafischen Funktionen eine unabhängige Realisierung zu erhalten, wurden Standardlösungen diskutiert. Dies war notwendig, da unterschiedliche Geräte, z.B. Plotter, in der Regel unterschiedliche Hard- und Softwareschnittstellen (Kommandos, Steuerleitungen, Kommunikationsprotokolle) besitzen. Verkäufer und Anwender erkannten, daß die Computergrafik–Industrie durch eine Überfülle von inkompatibler Hard- und Software gehandikapt ist. Die frühesten Vorschläge für Grafikstandards, **CORE** (1977) und **GKS** (Grafisches Kernsystem, 1978) wurden entwickelt, um die Bedürfnisse von **2D–Ausgaben** abzudecken. Sie befaßten sich mit Primitiven wie Linien, Markern und Polygonen, die mit Schraffuren und Farbe gefüllt sind. Sie haben auch in genügendem Umfang Textmöglichkeiten und können Eingaben von verschiedenen Geräten handhaben. Die Standards sind so ausgelegt, daß keine Geräteabhängigkeiten für den Anwendungsprogrammierer in Erscheinung treten. Sie stellen demnach eine sinnvolle Schnittstelle zwischen Anwenderprogrammen und der peripheren Grafikhardware dar.

10.4.1 PHIGS und OpenGL

Der Notwendigkeit, **dreidimensionale** Zeichnungen zu erstellen, widmet sich ein Standard mit Namen **PHIGS** (Programmer's Hierarchical Interactive Graphics System, 1984). PHIGS ist datenorientiert und widmet sich insbesondere dreidimensionalen Objekten wie Linien, Markern, Polygonen, Polyeder und Text. Es schließt die Konzeption eines hierarchischen Modells ein, wodurch dem Anwender gestattet wird, Manipulationen an den grafischen Daten vorzunehmen (z.B. in drei Dimensionen rotieren, die Farben modifizieren usw.), ohne die gesamte Datenbasis von Grund auf neu spezifizieren zu müssen.

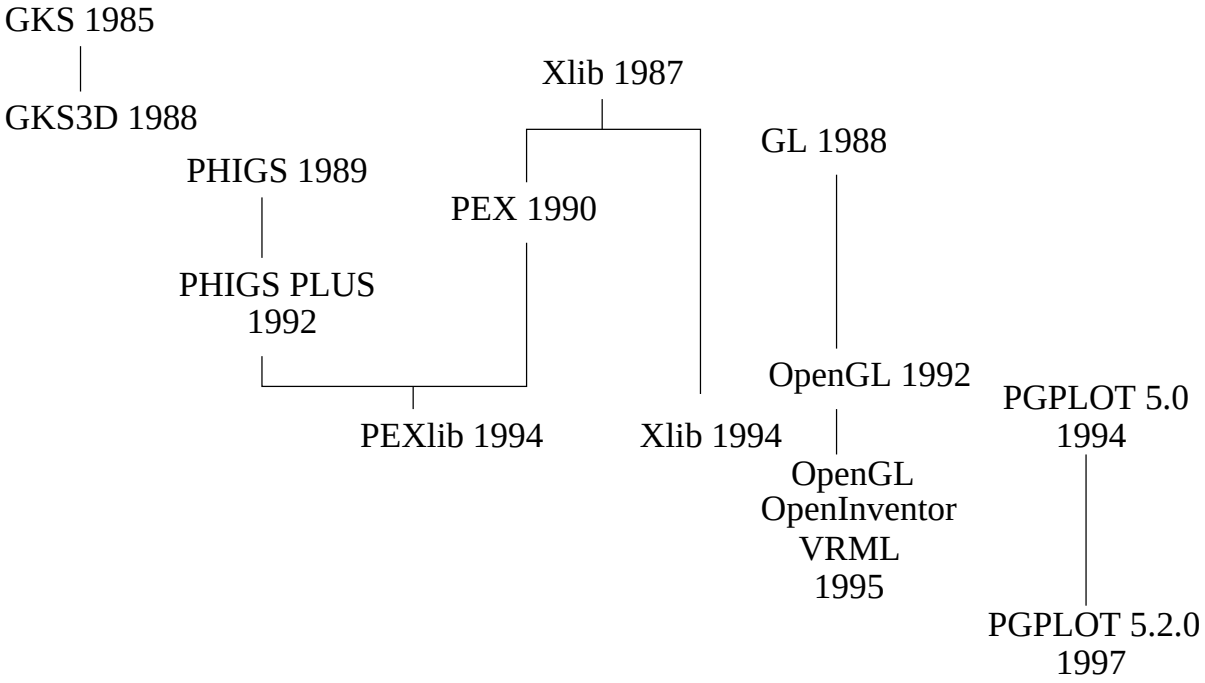


Abbildung 10.2: historische Entwicklung von Grafikstandards

Da insbesondere bei dreidimensionaler Darstellung von Animationen (z.B. Flugsimulator) der numerische Aufwand zur Berechnung enorm ansteigt und handelsübliche Prozessoren nicht in der Lage sind, diese Berechnungen in 'Echtzeit' durchzuführen, haben in den 80er Jahren Computerhersteller damit begonnen, spezielle Grafikhardware in ihre Rechner einzubauen; die 'Grafikbeschleuniger' wurden geboren. Aufgaben dieser Prozessoren sind z.B. die Berechnung sichtbarer bzw. verdeckter Flächen (Z–Buffer), das schnelle Füllen von Farbflächen (Rendering) und die Berechnung von Oberflächenstrukturen (Texture–Mapping). Viele dieser Aufgaben lassen sich sehr gut parallelisieren, so daß moderne Grafiksupercomputer neben einem oder mehrerer Numerikprozessoren auf ihren Grafikkarten (die von den Ausmaßen nichts mit den PC–Karten gemeinsam haben) eine Vielzahl von Prozessoren für die Grafikberechnung benutzen. Die Rechenleistung der Grafikprozessoren liegt dabei meist um Größenordungen über der Numerikleistung dieser Computer.

Um diese Spezialhardware effizient ansprechen zu können, wurde von der Firma Silicon Graphics (SGI) eine eigene Grafikbibliothek entwickelt, die sog. 'Graphics Library', GL. Diese Bibliothek enthält ähnlich wie GKS oder PHIGS Routinen und Funktionen zum Zeichnen von Grafikprimitiven. Im Unterschied zu den Softwareemulationen von z.B. GKS werden die Routinen direkt von der Grafikhardware ausgeführt, sofern sie vorhanden ist. Damit ändert sich die bisher dargestellte Struktur eines Grafikssytems.

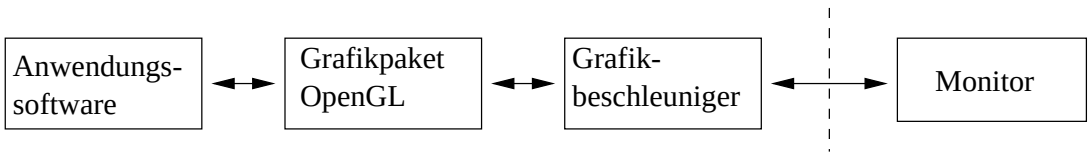


Abbildung 10.3: Struktur eines Grafikssystems mit OpenGL

Sind die entsprechenden Komponenten auf einem Computer nicht vorhanden, werden die Befehle wie bei GKS auf dem Numerikprozessor mit entspr. Performanceverlusten ausgeführt. Damit können Grafikprogramme auf der gesamten Hardwarepalette des Herstellers entwickelt und verwendet werden.

Der enorme Erfolg der Computer von SGI und der GL, der durch Filme wir 'Jurassic Park' (alle Computeranimationen wurden mit SGI–Rechnern durchgeführt) auch einem breiteren Publikum bekannt gemacht wurde, veranlaßte alle großen Computerhersteller (DEC, IBM, SUN, INTEL etc.) zusammen mit SGI einen erweiterten Standard zu entwickeln, die Open Graphics Library, OpenGL. Mit der OpenGL wurde ein Standard verabschiedet, der für den Softwareentwickler ein hardwareunabhängiges Werkzeug zur Entwicklung von Grafikprogrammen bereitstellt, daß auf Rechnern verschiedenster Hersteller lauffähig ist (Cross–Platform–Kompatibilität). Auf der anderen Seite können die Computerhersteller im Rahmen der Norm von OpenGL Spezialhardware für ihre Computer entwickeln und sich so trotz kompatibler Bibliotheken einen Wettbewerbsvorsprung gegenüber anderen Herstellern erarbeiten[1].

Für viele Anwendungsprogrammierer stellen die nur auf Grafikprimitiven (Punkt, Linie, Fläche etc.) und Attributen (Dicke, Farbe etc.) arbeitenden Bibliotheken wie OpenGL eine mächtige aber auch umständlich zu programmierende Umgebung dar, da Aufrufe, die komplexere Dinge wie z.B. das Zeichnen eines Koordinatensystems mit Skalierungen etc. eine Vielzahl von Bibliotheksaufrufen erforderlich machen. Daher bieten viele Hersteller zusätlich zu OpenGL weitere Bibliotheken an, die auf OpenGL basieren aber komplexe Grafikbefehle kennen um komlizierte Grafiken relativ einfach aufbauen zu können. Ein Beispiel hierfür ist die Bibliothek OpenInventor.

10.4.2 PGPLOT

Da die Fähigkeiten und der Programmieraufwand von OpenGL und OpenInventor weit über eine Einführung in die grafische Datenverarbeitung hinausgeht, wollen wir ein weiteres 'einfacheres' Paket vorstellen: PGPLOT. Bei PGPLOT handelt es sich um eine frei verfügbare Grafikbibliothek (**nicht** public domain) mit deren Hilfe einfache Plots für wissenschaftliche Zwecke erstellt werden können. Die Funktionalitäten von PGPLOT werden über Unterroutinen bzw. Funktionen angesprochen. Um beim Compilieren die Bibliothek in das ausführbare Programm einzubinden, muß bei den Micropoolrechnern folgender Befehl eingegeben werden:

```
<compiler-aufruf> myprog.f -o myprog -lpgplot -lX11 -L/usr/local/lib/pgplot -L/usr/X11/lib
```

wobei die Bibliothek 'X11' nur angegeben werden muß, falls eine graf. Ausgabe auf ein X–Fenster gewünscht ist.

Die Funktionsweise ist nun wie folgt:

A 10.1 Festlegen eines Ausgabemediums & des Zeichenbereichs

A 10.2 Festlegen, was geplottet werden soll

A 10.3 Beenden des Grafikprogramms

Zu 1.: PGPLOT kann Grafiken in verschiedenen Ausgabeformaten ausgeben, wobei in *interaktive* und *passive* Ausgabeformate unterschieden wird. Als interaktive Grafiken sind X-Fenster definiert, als passives Ausgabeformat steht im Micropool z.B. Postscript (Quer- und Hochformat) zur Verfügung. Das Ausgabemedium wird über die INTEGER–FUNCTION PGOPEN; z.B. definiert

```
IER = PGOPEN('/XWINDOW')
```

ein interaktives Ausgabefenster. Wenn man stattdessen

[1] Auch für Linux–Computer gibt es inzwischen OpenGL, wobei im Micropool des HFI eine GNU Version namens Mesa installiert ist. Mesa führt alle Befehle unabhängig von der installierten Grafikkarte 'in Software' auf dem Numerikprozessor aus

```
      IER = PGOPEN('?')
```

in sein Programm schreibt, werden die auf dem jeweiligen System konfigurierten Ausgabemedien angezeigt, und man wird aufgefordert, ein entsprechendes Ausgabeformat einzugeben. Ist der Wert der Funktion ≥ 0 konnte das Medium erfolgreich ausgewählt werden, Werte < 0 deuten auf einen Fehler hin.

Der Zeichenbereich wird mit der Unterroutine PGENV definiert:

```
      CALL PGENV(XMIN,XMAX,YMIN,YMAX,JUST,AXIS)
```

XMIN ,XMAX und YMIN, YMAX stellen den darzustellenden Zeichenbereich dar, während JUST und AXIS Parameter für die Skalierung und das Zeichnen von Rahmen sind (vgl. Doku auf unserer WWW-Seite).

Zu 2.: Es steht eine Vielzahl von Programmen zum Plotten von Daten zur Verfügung. Dazu gehören u.a. die Unterroutinen

- PGLAB (Achsenbeschriftung)
- PGLINE (Ziehen von Linien)
- PGPT (Zeichnen von Punkten)
- PGTEXT (Text in der Grafik)
- PGIDEN (user id und Datum in der Grafik)

Ausserdem können Plotparameter verändert werden, z.B.:

- PGSLS (Art der Linien)
- PGSLW (Strichstärke)
- PGSCI (Farbe)

Somit können z.B in einem Diagramm verschiedene Kurven mit versch. Kurventypen dargestellt werden:

```
      ...
C     Stelle durchgezogene Linie ein:
      CALL PGSLS(1)
C     Plotte Linie
      CALL PGLINE(..)
C     Stelle gestrichelte Linie ein:
      CALL PGSLS(2)
      CALL PGLINE(..)
      ..
```

Insgesamt stehen über 120 Routinen zum Plotten und Manipulieren von Grafiken zur Verfügung (vgl. doku).

Eine interessanter Aspekt von PGPLOT ist die Möglichkeit *interaktive* Grafiken zu erzeugen; d.h. das Grafikprogramm reagiert auf Maus- oder Tastatureingaben des Benutzers. Hauptelement der Interaktionen ist die Funktion PGCURS, mit der Cursorabfragen auf der Grafikoberfläche gemacht werden können. Auf unserer WWW Seite haben wir als Beispiel für eine interaktive Grafik das Auswahlmenü der Programmieraufgabe gelegt.

10.5 PostScript

Die Sprache **PostScript** ist eine Programmiersprache, die entworfen wurde, um eine Beschreibung nahezu jeder gewünschten Seite an einen Drucker übermitteln zu können. Sie wurde als moderner Druckstandard entworfen und im Jahre 1985 von der Firma Adobe Systems veröffentlicht. Ihr Ursprung reicht bis ins Jahr 1972 zurück.

PostScript besitzt ein weites Spektrum von grafischen Operatoren, die beliebig kombiniert werden können. Sie enthält Variablen und gestattet die Kombination von Operatoren zu komplexeren Prozeduren und Funktionen.

Seitenbeschreibungen in PostScript sind Programme, die von einem Interpreter verarbeitet werden. PostScript–Programme werden normalerweise von Anwendungsprogrammen (Grafik–Programmen, Textverarbeitungssystemen) erzeugt, die auf anderen Computern laufen können. Man kann PostScript als Allzweck–Programmiersprache mit mächtigen eingebauten Grafik–Primitiven bezeichnen; genauso gut trifft die Definition „Seitenbeschreibungssprache mit Merkmalen einer Programmiersprache“ zu. Ein Bild von der Mächtigkeit und Flexibilität von PostScript sollen die folgenden Absätze vermitteln (nach [2, 7/88]).

Die Eigenschaften einer strukturierten Programmiersprache besitzt PostScript durch das Vorhandensein von Kontrollstrukturen wie Verzweigungen, Schleifen und Unterprogrammen in seiner Syntax. Diese Syntax ist nur minimal festgelegt und daher ungeheuer flexibel und anpassungsfähig an unterschiedlichste Anforderungen. PostScript ist wie erwähnt eine Interpretersprache und verwaltet Objekte. Bei den Objekten wird nicht zwischen Befehlen (Operatoren) und Daten (Operanden) unterschieden. Jedes Objekt wird vom Interpreter grundsätzlich sofort ausgeführt, und zwar nicht nur Operatoren, sondern auch Operanden. Das Ausführungsverhalten eines Operanden besteht nur darin, sich selbst auf einen der für ihn bestimmten Stapel (stacks) abzulegen. Andere Objekte (Operatoren) nehmen sich benötigte Parameter von diesem Stapel herunter und legen dort Ergebnisse ab, oder aber sie haben 'Seiteneffekte' wie z.B. den Ausdruck einer Textseite.

PostScript–Objekte haben einen Datentyp (z.B. integer, real, string, array, dictionary (Symboltabelle), fontID (Zeichensatz–Datenstruktur) und verschiedene Attribute (z.B. executable (als Befehl ausführbar), literal (nur als Operand nutzbar), read only (nicht veränderbar)).

Zeichensätze sind bei PostScript grundsätzlich vektoriell, also mit Polygonzügen, realisiert. Sie lassen sich daher stufenlos vergrößern und verkleinern, es gibt also keine quantisierbaren Sprünge, wie sie irgendein vorgegebenes Raster erzwingt.

Das **grafische Konstruktionsprinzip** geht von einer weißen Seite aus, auf die mit deckenden Farben grafische Objekte gemalt werden, d.h. neu gemalte Objekte können bereits existierende überdecken. Ein sichtbares Ergebnis erhält man erst, wenn nach dem kompletten Aufbau der Seite der Befehl 'showpage' ausgeführt wird.

Das Koordinatensystem und sein Ursprung können beliebig skaliert (vergrößert/verkleinert, Operator 'scale'), gedreht ('rotate') und verschoben ('translate') werden. Erst durch die Skalierung geben später angegebene Koordinaten und Größenangaben einen Sinn. Zu Beginn liegt der Ursprung in der linken unteren Ecke, die Maßeinheit ist ein 'Pica Point' (1/72 Zoll).

Für ein grafisches Objekt ist zunächst dessen Umriß, der 'Pfad' ('path') zu definieren. Das geschieht mit (den wohl selbsterklärenden) Operatoren wie 'newpath', 'moveto', 'lineto', 'curveto' und 'closepath'. Bevor der so definierte Weg sichtbar gemacht werden kann, müssen noch Grafik-Attribute wie Liniendicke, Farbe, Füllmuster und Zeichensätze definiert worden sein. Erst nach Definition von Pfad und Attributen kommt eine Zeichenoperation zum Einsatz. Dabei führt 'fill' zum Darstellen der Fläche mit dem durch den Pfad beschriebenen Umriß unter Berücksichtigung von aktueller Farbe und momentanem Füllmuster. 'stroke' zeichnet eine Linie entlang des Pfades mit der aktuellen Linienstärke. Mit 'image' lassen sich Rastergrafiken (z.B. Fotos) und mit 'show' Texte darstellen.

Daneben läßt sich mit 'clip' ein Pfad erzeugen, der die Grenzen der nutzbaren Zeichenfläche definiert.

Objekte, die diese Grenze überschreiten, werden abgeschnitten. Clip–Pfade können beliebige Formen (wie etwa auch die Umrisse von Buchstaben) haben, die Gestaltungsmöglichkeiten sind nahezu unbegrenzt. Auch die Grundlinie für Schriften kann damit festgelegt werden, womit die eigenwilligsten Entwürfe realisierbar sind.

Schriften und Zeichensätze und damit verbundene Operatoren nehmen bei PostScript breiten Raum ein. Zeichen sind ebenfalls grafische Objekte, die in speziellen Font–Dictionaries abgelegt sind. In jedem Font–Dictionary ist für jeden Zeichencode eine eigene Prozedur (Operator) abgelegt, welche das Zeichen darstellt.

Der PostScript–Befehlssatz umfaßt Operatoren zur Manipulation des Parameterstapels, arithmetische Operatoren und transzendente Funktionen, Operatoren für Felder, Dictionaries und Zeichenketten, Vergleichs– und Bit–Operatoren, Operatoren zur Programm–Ablaufsteuerung und zur Manipulation des 'Graphic State' (Zusammenfassung aller Attribute, Pfade, Koordinatensysteme etc.), Operatoren für Typkonvertierung und Attribut–Funktionen, Koordinatensystem–, Datei– und Pfad–Operatoren, Zeichen–, Ausgabe– und Geräte–Befehle sowie Zeichensatz–Operatoren.

Wer sich genauer für PostScript interessiert, sei auf die Literatur verwiesen (z.B. [13], [17], [2, 7/88]). Ein einfaches Beispiel in Abb. 10.4 soll diesen Abschnitt über PostScript beschließen.

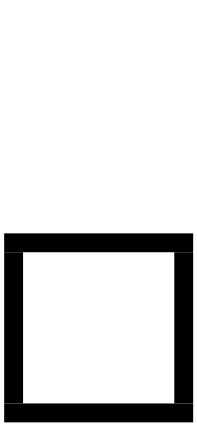


Abbildung 10.4: Output des PostScript–Programm

```
%define a box procedure
/box
{
 newpath
    0 0 moveto
    1 0 lineto
    1 1 lineto
    0 1 lineto
    closepath
} def

72 72 scale             %scale coordinate system so that 1 unit = 1 inch

/Times-Roman findfont   %put this font at the top of the stack
0.2 scalefont           %scale this font to 1/5 th inch
setfont                 %and make this the current font
```

```
1 8 moveto              %move to new position
(Figure 10.7 - Output of the example PostScript program) show


1 1 translate           %translate origin 1 inch up and to the right
0.1 setlinewidth        %set the line width to 1/10 inch
box stroke

2 2 translate           %translate origin by 2 inches in x and y
0.0 setgray             % set gray scale for black
box fill

showpage                %output the described page
```

10.5.1 Portable Document Format (PDF)

10.5.1.1 Überblick

PDF ist der offene de-facto Standard für den Austausch von elektronischen Dokumenten. Adobe PDF ist ein universelles Dateiformat, das alle Schriften, Formatierungen, Farben und Grafiken jedes Ausgangsdokuments beibehält, unabhängig von der Anwendung und der Plattform, die zur Erstellung verwendet wurden. PDF-Dateien sind kompakt und können gesperrt, angezeigt, durchgeblättert und gedruckt werden, jedes beliebige Dokument kann in PDF konvertiert werden. Zum Betrachten und Drucken von PDF-Dateien ist der **Acrobat Reader** von Adobe notwendig, der frei verfügbar ist.

Probleme, die z.B. entstehen, wenn der Benutzer eine Datei öffnen möchte ohne die Anwendung zu besitzen, mit der sie erstellt wurde, entfallen hier. Auch das Verlorengehen von Formatierungen, Schriften und Grafiken durch Inkompatibilität der Plattformen, Software oder Versionsunterschiede der Software ist bei PDF nicht möglich. PDF-Dateien sehen immer so aus und werden so gedruckt, wie sie erstellt wurden, da das PDF-Dokument alle für die Darstellung und Druckausgabe benötigten Fontinformationen enthält.

Als Dateiformat kann PDF als eine Weiterentwicklung der Druckerbeschreibungssprache PostScript zu einem Format für elektronische Dokumente gesehen werden. PDF bietet alle Layout-Möglichkeiten eines PostScript-Dokuments. PostScript ist bei der Erzeugung von PDF-Dokumenten das zentrale Zwischenformat, zumeist durch einen PostScript-Druckertreiber wird erst die PostScript-Datei und daraus die z.B. mit dem **Acrobat Distiller** die PDF-Datei erzeugt. Dabei spielt die Anwendung, mit der die ursprüngliche Datei erstellt wurde, keine Rolle.

10.5.1.2 Acrobat

Das kostenpflichtige Programmsystem **Acrobat** der Firma **Adobe** dient zur Bearbeitung und Erzeugung von PDF-Dokumenten. Hiermit können zusätzliche Funktionen wie z.B. Hypertext-Links, Multimedia-Elemente, Lesezeichen oder Felder für Online-Formulare eingefügt werden. Prinzipiell existieren aber auch andere Tools zur Erzeugung von PDF-Dokumenten aus PostScript oder auch z.B. direkt aus TeX-Dateien.

10.5.1.3 Anwendungsbereiche

Mit seinen vielfältigen Möglichkeiten, der freien Verfügbarkeit des **Acrobat Reader** und dessen problemloser Integration in die meisten Browsertypen ergeben sich gerade im Bereich des WWW und des Datenaustauschs viele Anwendungsmöglichkeiten: Die Verteilung von Dokumenten dessen Layout wichtig

ist im WWW, Dokumentationen, z.B. elektronische Handbücher oder auch die Weitergabe von Dokumenten an Druckereien und die Archivierung von Dokumenten in elektronischer Form.

10.6 Zusammenfassung

Die Grafische Datenverarbeitung befaßt sich mit der rechnergestützten Erzeugung, Speicherung, Darstellung und Manipulation von Bildern. Hierfür verwendete Grafiksysteme bestehen aus Software (Programme) und den Geräten (Hardware). Um die aufwendige Grafik–Software portabel und damit unabhängig von bestimmten Geräten und Rechnern zu machen, haben sich Grafik–Standards durchgesetzt.

Die ersten genormten Standards wie das Grafische Kern System GKS waren auf zweidimensionale Ausgabe zugeschnitten. Es gibt Primitive, wie Linien, Marker und Polygone, die mit Farbe und Schraffuren gefüllt sind, sowie Textmöglichkeiten und Eingabemöglichkeiten von verschiedenen Geräten. Für dreidimensionale Grafiken ist PHIGS entworfen worden. Es ist datenorientiert und behandelt dreidimensionale Objekte wie Linien, Polygone, Text und Polyeder. Ein hierarchisches Objekt–Modell erlaubt die Manipulation an grafischen Daten ohne Neudefinition der grafischen Datenbasis.

Seit der Einfuehrung 1992 hat sich OpenGL als Standard für die Entwicklung und Ausgabe von 3D–Grafiken etabliert. Spezielle Grafikhardware kann heute OpenGL–Befehle direkt in der Hardware ausführen und erzielt somit eine signifikante Beschleunigung der Darstellung und damit auch eine Erweiterungen der Anwendungsmöglichkeiten.

Dia Anbindung der genormten Grafikstandards über geräteunabhängige Schnittstellen an reale Ein–Ausgabegeräte geschieht über Treiberprogramme, die die neutrale Grafik–Information gerätespezifisch umsetzen.

Ein weithin akzeptierter de–facto–Standard ist — vornehmlich für Laserdrucker — die mächtige Seitenbeschreibungssprache PostScript.

10.7 Fragen und Übungen zu Kapitel 10

F 10.1 Was ist Grafische Datenverarbeitung (GDV)? Was ist ein Grafiksystem?

F 10.2 Nenne Anwendungsgebiete der GDV.

F 10.3 Nenne Beispiele für genormte Grafikstandards.

F 10.4 Nenne Beispiele für de–facto Grafikstandards.

F 10.5 Welche Vor- und Nachteile haben genormte gegenüber de–facto Standards?

F 10.6 Was ist PostScript?

F 10.7 Wie kann man PostScript–Programme erzeugen?

10.8 Antworten zu den Fragen und Übungen zu Kapitel 10

A 10.1 Die Grafische Datenverarbeitung befaßt sich mit der Generierung, Speicherung, Darstellung und Manipulation von Bildern. GDV geschieht mit Grafiksystemen, das sind Anwendungsprogramme zur Erfüllung der Aufgaben und die dazugehörenden Rechner und Ein–/ Ausgabe–Geräte.

A 10.2 CAD, CAE, CAM

Visualisierung

Architektur

Entwurf

Animation

Medizin

A 10.3 GKS, PHIGS, OpenGL

A 10.4 Postscript

A 10.5 genormte Standards:

- Vorteil:
  - herstellerunabhängig
  - hardwareunabhängig
  - daher portable Grafikprogramme
- Nachteil:
  - langwierige Vorbereitung der Standards
  - keine Musterimplementierung
  - keine Möglichkeit zur Ausnutzung spezieller Hardware

A 10.6 Eine sehr mächtige Seitenbeschreibungssprache der Firma Adobe, die sich zum de–facto Standard für Laserdrucker–Ausgabe entwickelt hat. Programmiersprache mit vielfältigen mächtigen Grafikmöglichkeiten.

A 10.7 Mit einem Editor oder (üblicherweise) als Output von Anwendungsprogrammen, z.B. Textverarbeitungsprogramme oder Grafikpakete.

Kapitel 11

Textverarbeitung

11.1 Allgemeine Einführung

Im Zuge der weiten Verbreitung von Personal Computern (PC) kam erstmals eine nennenswerte Zahl von Laien mit dem Computer als Werkzeug in Berührung. Neben Tabellenkalkulationsprogrammen war (und ist) dann die erste Anwendung ein Textverarbeitungsprogramm. Im Vergleich zu einer Schreibmaschine liegen die Vorteile auf der Hand:

- Leichte Korrektur von Tippfehlern
- Texte können überarbeitet, aktualisiert, erweitert und neu kombiniert werden
- vielfältige Gestaltungsmöglichkeiten

Um die Möglichkeiten der modernen Textverarbeitung beurteilen zu können, müssen wir zuerst die folgende Unterscheidung machen:

- „intelligente“ Schreibmaschinen
- Computer mit Textverarbeitungsprogramm

Die Hersteller von Schreibmaschinen versuch(t)en ihre Geräte durch Einbau von immer mehr Elektronik zum Computer konkurrenzfähig zu halten. In einem ersten Schritt wurden die Schreibmaschinen mit kleinen Flüssigkristall–Anzeigen (LCD) ausgestattet. Heute gibt es bereits Geräte mit Monitor und Diskettenlaufwerk. Nachteilig bleibt jedoch, daß diese Geräte nur für Textverarbeitung zu benutzen sind.

Im Zusammenhang mit Textverarbeitung auf dem Computer müssen wir folgende Begriffe unterscheiden:

- Editoren
- Textverarbeitungsprogramme
- Textformatierer
- Desktop–Publishing (DTP)

11.2 Formen der Textverarbeitung

11.2.1 Der Editor

Editoren dienen zum Erstellen und Bearbeiten von ASCII–Dateien. Merkmale von Editoren wurden bereits in Kapitel 3 vorgestellt.

11.2.2 Das Textverarbeitungsprogramm

Textverarbeitungsprogramme wollen wir solche Programme nennen, die die Funktionen

- Texteingabe
- Formatierung
- Ausgabe auf Drucker

in einem Programm (in einer Benutzeroberfläche) vereinigen und im allgemeinen auf **ein** bestimmtes Betriebssystem beschränkt sind. Das Grundübel der Textverarbeitung ist folgendes: Beim Abspeichern z.B. eines Briefes muß die Datei auch die Informationen über Kursivschrift, Unterstreichungen und Absätze etc. beinhalten. Es gibt leider keinerlei Normen, wie dies zu geschehen hat. Folglich benutzt jede Textverarbeitung ihr eigenes Format, so daß Textdateien in der Regel nur von dem zugehörigen Textverarbeitungsprogramm gelesen werden können. Es haben sich aber Formate von bestimmten Textverarbeitungsprogrammen als Quasi–Standards durchgesetzt. So können Dateien von z.B. MS–Word, WordPerfect etc. von einigen anderen Programmen eingelesen und weiterverarbeitet werden.

Aufgrund der Zusammenfassung von Eingabe, Formatierung und Ausgabe des Textes sind Textverarbeitungsprogramme sehr beliebt und verbreitet. Über den ständig wachsenden Markt der Textverarbeitungsprogramme informiert man sich am besten aus einschlägigen Computerzeitschriften, z.B. [5], [2].

11.2.3 Der Textformatierer

Textformatierer sind Programme, die einen Text, welcher z.B. mit einem Editor erstellt wurde, in einer bestimmten Form umarbeiten. Das Resultat muß im allgemeinen zur Ausgabe auf einem Drucker von einem speziellen Programm, dem Druckertreiber, ein weiteres Mal umgewandelt werden. Diese Trennung von Texteingabe, Formatierung und Ausgabe stammt aus dem Bereich der Großrechner bzw. dem Druckereigewerbe.

11.2.4 Das Desktop–Publishing–Werkzeug

Desktop–Publishing (DTP) steht für die Erzeugung kompletter Dokumente (Broschüren, Zeitungen etc.) an *einem* Schreibtisch. Hier steht die Einbindung von Grafiken und Photos in mehrspaltigen Text im Vordergrund. Da für echtes DTP außerdem übergroße Bildschirme und schnelle Rechner mit großer Kapazität (RAM, Platte) benötigt werden, sind diese Lösungen recht teuer und nicht portabel. DTP–Anwendungen sind nicht einfach zu bedienen und setzen im Grunde genommen professionelle Kenntnisse voraus. Anders gesagt [4]:

> „DTP–Programme ... verleiten zum Spielen, sie erlauben jedem alles (Schriftarten, Satzspiegel ...) und genau so sehen hinterher auch die Schriftstücke aus.“

11.3 Auswahl einer Textverarbeitung

Um nun zu entscheiden, welche Form von Textverarbeitung im konkreten Fall geeignet ist, muß man die folgenden Punkte bedenken bzw. abwägen:

1. **WYSIWYG**: (engl. *what you see is what you get*) Wird schon bei Eingabe des Textes dieser so dargestellt, wie er später auf dem Papier aussieht? Wenn dies nicht der Fall ist, sollte zumindest ein **Preview** möglich sein, d.h. ein spezielles Programm erlaubt die Darstellung des Textes mit allen Zeichensätzen und dem kompletten Layout auf dem Bildschirm.
2. Welche **Hardware** ist nötig?
3. Ist das Programm **netzwerkfähig**?
4. Ist das **Einbinden von Grafiken** möglich, und wenn ja, in welchen Formaten?
5. Hat das Programm **grafische Grundfunktionen**?
6. Erlaubt das Programm die Darstellung von **mathematischen, chemischen o.a. Formeln**?
7. Was **kostet** das Programm?
8. Wie **verbreitet** ist das Programm?
9. Stehen die notwendigen **Druckertreiber** zur Verfügung?
10. Lassen sich einmal erstellte Formeln oder Strukturen als **Makros** oder in einer Bibliothek abspeichern?
11. Kann das Programm **Proportionalschrift**, d.h. werden die Buchstaben `m` und `i` verschieden breit dargestellt?
12. Welche **Satzmöglichkeiten** (Blocksatz, Zentriert, Spaltensatz etc.) gibt es?
13. Gibt es eine **grafische Benutzeroberfläche** inkl. einer **Hilfe–Funktion**?
14. Gibt es Schnittstellen zu **Datenbank-** bzw. **Tabellenkalkulationsprogrammen**?
15. Gibt es eine automatische **Rechtschreibprüfung**?
16. Wird **automatisches Trennen** durchgeführt?
17. Kann das Programm mit **verschiedenen Sprachen** umgehen?
18. Stehen Mechanismen für das automatische Anlegen von **Fußnoten, Stichwortregister und Inhaltsverzeichnis** zur Verfügung?
19. Gibt es einen **Thesaurus**, d.h. Vorschläge von sinnverwandten Wörtern, um den Stil von Texten zu verbessern?

Die Auswahl einer Textverarbeitung ist daher mit der Auswahl einer Programmiersprache vergleichbar. Die Anzahl der Textverarbeitungsprogramme ist jedoch erheblich größer als die der höheren Programmiersprachen!

11.4 Textverarbeitung unter UNIX

Das Betriebssystem UNIX findet immer mehr Verbreitung. Es soll deshalb erwähnt werden, daß UNIX mächtige Werkzeuge zum Formatieren von Texten zur Verfügung stellt.

Ein weitverbreitetes Formatierprogramm ist LaTeX[1], zu dem hier eine kurze Einführung gegeben werden soll. Ausführlichere und detailliertere Informationen sind im Internet bzw. entsprechender Literatur z.B. [9], [14] und [8][2] zu finden.

11.4.1 LaTeX

11.4.1.1 Was ist LaTeX ?

LaTeX ist ein Makropaket, das auf TeX[3] aufsetzt, siehe auch [9]. TeX ist ein Computersatzprogramm, das in den 70er Jahren von dem amerikanischen Mathematiker Donald Knuth an der Stanford University zum Setzen mathematischer Formeln entwickelt wurde [7]. TeX kommt vom griechischen $\tau\epsilon\chi$, welches sowohl Technik als auch Kunst bedeutet.

Bei Benutzung von TeX ohne LaTeX muß sich der Benutzer um das Layout seines Textes selber kümmern. Das Makropaket LaTeX erzeugt selbständig ein Layout, das auf den Erfahrungen professioneller Schriftsetzer beruht und somit im Normalfall besser ist als eigene Versuche. Es empfiehlt sich also generell die Benutzung von LaTeX.

Zwischen weit verbreiteten Textverarbeitungsprogrammen wie MS–Word und WordPerfect und teuren, leistungsfähigen DTP–Paketen wie PageMaker oder Framemaker nimmt LaTeX eine Mittelstellung ein. Vom Leistungsumfang übersteigt es die PC–Programme, vom Komfort reicht es nicht an die DTP–Programme heran. Das Besondere an LaTeX ist, daß sowohl Quellcode als auch Binaries (ausführbare Programme) in der Public Domain sind, also kostenlos verfügbar sind. Obwohl auch kommerzielle Versionen im Handel sind, unterscheidet sich LaTeX an dieser Stelle deutlich von den anderen oben genannten Produkten. Zum Thema LaTeX gibt es auch deutsche Bücher, z.B. [14], [8].

Zur Verbreitung von LaTeX ist zu sagen, daß es an fast allen Hochschulen zu finden ist (auch an der TU–Berlin) und dort auf Rechnern aller Größenordnungen. Für viele Fachrichtungen gibt es Makros, die auf spezielle Bedürfnisse eingehen, z.B. für Musiker, Physiker, Chemiker oder Elektrotechniker. Fachverlage, wie Springer, oder Fachzeitschriften, wie Physical Review, nehmen von ihren Autoren LaTeX–formatierte Manusskripte entgegen.

LaTeX ist kein WYSIWYG–Programm, das heißt der Anwender sieht bei der Eingabe des Textes am Bildschirm nicht, wie das Elaborat gedruckt aussehen wird. Um das Grundprinzip und die Leistungsfähigkeit zu verdeutlichen, ein Beispiel:

Die Eingabe

```
Ein {\LARGE Computer
kann {\bf alles}},
aber
sonst {\huge \sl nichts!!!}
```

ergibt folgende Ausgabe

Ein Computer kann **alles**, aber sonst *nichts!!!*

[1] LaTeX ist auch für das Betriebssystem Windows verfügbar
[2] Dieses Buch war auch hilfreicher Begleiter während der Erstellung dieses Vorlesungsskriptes
[3] Aussprache wie die erste Silbe im Wort „Technik“

11.4.1.2 Arbeitsweise

Die Erstellung eines Textes mit LaTeX ist ein mehrstufiger Prozeß. Zunächst wird mit einem beliebigen Editor der Rohtext erstellt. Ein wesentlicher Faktor der Systemunabhängigkeit ist, daß der Rohtext eine ASCII–7–Bit–Datei ist, die man problemlos von Rechner zu Rechner transferieren kann, sei es auf Disketten oder per e–mail. Dieser Rohtext besteht aus einem Vorspann (Preamble) und einem Textteil (engl. *body*). Im Vorspann werden generelle Einstellungen wie Dokumentenstil, Seitenlänge, -breite und Schriftart festlegt, wobei einige dieser Attribute im laufenden Text auch wieder verändert werden können. Darüber hinaus kann man zum Beispiel mit `\author{..}` und `\title{..}` Author und Titel des Dokumentes angeben. Der Textteil wird durch die Befehle `\begin{document} ... \end{document}` eingeschlossen.

Im zweiten Schritt wird der Rohtext zu einem geräteunabhängigen Binärfile compiliert (DVI — engl. *device independent*), wobei Makros, wie LaTeX, verarbeitet und Fonts[4] eingebaut werden. Dieses DVI–File kann dann entweder mit einem Previewer gesichtet oder zu einer Gerätedatei für einen Drucker verarbeitet werden.

11.4.1.3 Steueranweisungen

Neben dem eigentlichen Text enthält ein LaTeX–File die zugehörigen Steuerungsanweisungen. Der Backslash \ leitet die Anweisungen ein. Dann folgt der Anweisungsname und ein abschließendes Leerzeichen bzw. ein anderes Sonderzeichen. LaTeX unterscheidet große und kleine Buchstaben. Eine LaTeX–Anweisung hat die Form: `\Befehl` [optional] { zwingend }.

Es gibt auch noch Anweisungen, die aus einem \ und einem Sonderzeichen oder einer Ziffer bestehen. Ein anderes wichtiges Zeichen ist das %–Zeichen. Erscheint ein %–Zeichen in einer Zeile, so wird der Rest der Zeile nicht mehr interpretiert (Kommentare). Die geschweiften Klammern { und } sind sogenannte Begrenzer. Sie werden benutzt, um einerseits die Argumente von Steuerungsanweisungen und Makros zu klammern, andererseits dienen sie auch zur Definition von lokalen Gültigkeitsbereichen.

11.4.1.3.1 Einige wichtige Steueranweisungen:

- Besondere Zeichen (bei Verwendung des Makropaketes `german`)
  - `"a` bzw. `"A` → ä bzw. Ä
  - `"s` → ß
- Gliederungsbefehle
  - `\maketitle` erzeugt eine Titelseite
  - `\chapter{Kapitel}`
  - `\section{"Uberschrift}`
  - `\subsection{Unter"uberschrift}`
  - `\subsubsection{Unterunter"uberschrift}`
  - `\paragraph{Absatz"uberschrift}`
  - `\tableofcontents` erzeugt ein Inhaltsverzeichnis
- Aufzählungen
  - Aufzählung mit vorangestellten Punkten, Anstrichen
    - `\begin{itemize} ... \item ... \item ... \end{itemize}`

[4] Ein Font ist ein Schrifttyp

- numerierte Aufzählung
  - `\begin{enumerate} ... \item ... \item ... \end{enumerate}`
- Aufzählungen lassen beliebig ineinander verschachteln

11.4.1.4 Verschiedene Schrifttypen

Ein besonderer Vorteil, den ein Satzsystem bietet, ist die Auswahl verschiedenster Schrifttypen in verschiedenen Schriftgrößen. Diese verschiedenen Schrifttypen werden „Fonts“ genannt (fontcase= Setzkasten). Folgende Zeichensätze können direkt benutzt werden:

- mit `\rm` wird in roman umgeschaltet
- **mit `\bf` wird in boldface umgeschaltet**
- *mit `\it` wird in italic umgeschaltet*
- *mit `\sl` wird in slanted umgeschaltet*
- `mit \tt wird in typewriter umgeschaltet`
- mit `\underline{..}` wird unterstrichen ausgegeben

Diese Fonts können noch in vergrößerter oder verkleinerter Version ausgegeben werden.

- mit `\tiny` wird sehrklein ausgegeben
- mit `\small` wird klein ausgegeben
- mit `\footnotesize` wird etwas größer ausgegeben
- mit `\normalsize` wird normal ausgegeben
- mit `\large` wird groß ausgegeben
- mit `\Large` wird größer ausgegeben
- mit `\LARGE` wird sehr groß ausgegeben
- mit `\huge` wird ziemlich groß ausgegeben
- mit `\Huge` wird ganz groß ausgegeben

Außerdem gibt es normalerweise noch sehr viel mehr Fonts, die auf einer Festplatte mehrere Megabyte belegen können.

11.4.1.5 Erstellen von Tabellen

Tabellen werden innerhalb einer sogenannten Tabellenumgebung erstellt. Folgendes Beispiel erzeugt eine drei–spaltige Tabelle, wobei die erste Spalte zentriert, die zweite Spalte linksbündig und dir dritte Spalte rechtsbündig formatiert sind. Der Befehl `\hline` dient dazu eine horizontale Linie zu erzeugen. Die vertikalen Begrenzungslinien werden mit den Pipes im Kopf der Tabellendefinition veranlaßt. Das & dient zum Beenden einer Spalte und der \\ zum Beenden einer Zeile.

```
\begin{tabular}{|c|l|r|}
\hline
Spalte 1 & Spalte 2 & Spalte 3 \\
Zeile 2 & Zeile 2 & Zeile 2 \\
\hline
Spalte 1, Zeile 3 & \dots & \dots \\
\hline
\end{tabular}
```

$\Rightarrow$

Spalte 1 Zeile 2	Spalte 2 Zeile 2	Spalte 3 Zeile 2
Spalte 1, Zeile 3	...	...

11.4.1.6 Mathematischer Modus

11.4.1.6.1 Die Begrenzer Um den mathematischen Modus ein- und auszuschalten, muß ein $ bzw. $$ gesetzt werden. Das $$ begrenzt einen besonders hervorgehobenen mathematischen Teil. Das $ kann auch mitten im Text angewandt werden.

Beispiel: `$$ \sum_{n=1}^m n+n^2$$` erzeugt

$$\sum_{n=1}^{m} n + n^2$$

während `$ \sum_{n-1}^m n+n^2$` folgendes erzeugt: $\sum_{n-1}^{m} n + n^2$

11.4.1.6.2 Griechische Buchstaben Griechische Buchstaben werden nur im mathematischen Modus dargestellt, also mit den Begrenzern $ oder $$

α `\alpha`	ι `\iota`	ϱ `\varrho`
β `\beta`	κ `\kappa`	σ `\sigma`
γ `\gamma`	λ `\lambda`	ς `\varsigma`
δ `\delta`	μ `\mu`	τ `\tau`
ϵ `\epsilon`	ν `\nu`	υ `\upsilon`
ε `\varepsilon`	ξ `\xi`	ϕ `\phi`
ζ `\zeta`	o `o`	φ `\varphi`
η `\eta`	π `\pi`	χ `\chi`
θ `\theta`	ϖ `\varpi`	ψ `\psi`
ϑ `\vartheta`	ρ `\rho`	ω `\omega`

Die meisten großen griechischen Buchstaben werden wie die kleinen dargestellt, nur mit einem großen Anfangsbuchstaben, z.B. `\Omega` für Ω. Ausnahmen bilden diejenigen, die wie Lateinische aussehen, z.B. Alpha, Beta, Epsilon. Diese werden als Lateinische dargestellt und zwar im „roman–font“.

11.4.1.6.3 Exponenten und Indizes Exponenten werden mit dem Zeichen ^ dargestellt, Indizes mit dem Zeichen _ . Zum Beispiel erzeugt `$x^{n+1}_i$` das Ergebnis x_i^{n+1}.

11.4.1.6.4 Wurzeln Quadratwurzeln werden mit `\sqrt`, n-te Wurzeln mit `\root n \of` eingegeben.

Beispiel: `$$\root n-1 \of {x^n + y^{n-1}}$$` erzeugt

$$\sqrt[n-1]{x^n + y^{n-1}}$$

11.4.1.6.5 Mathematische Operatoren Wenn die Operatoren obere und untere Grenzen haben, so erhalten sie diese mit den Zeichen „_“ (untere Grenze) und „^ “ (obere Grenze). Wichtige Beispiele:

```
$$\sum_{n=1}^m a^n$$
```

$$\sum_{n=1}^{m} a^n$$

```
$$\prod_{i=1}^n {(i+1) \over (i+1)!}$$
```

$$\prod_{i=1}^{n} \frac{(i+1)}{(i+1)!}$$

```
$$\int_{n-b}^a{h_2e^{-i\omega t}}dt$$
```

$$\int_{n-b}^{a} h_2 e^{-i\omega t} dt$$

11.4.1.7 Beispiel

TEX in der Grundversion verfügt über etwa 900 Befehle[5]. Hinzu kommen die Makros von LATEX. Im Rahmen dieses Scriptes soll deshalb nur ein erster Eindruck von Wirkungsweise und Möglichkeiten von LATEX gegeben werden. Weitergehende Informationen sind der angegebenen Literatur zu entnehmen. Wir beschränken uns deshalb auf ein abschließendes Beispiel: Der folgende Text

```
\documentclass[a4paper]{book}
\usepackage{german,a4}
\begin{document}
%   Das ist das Quellfile, hinter "%" stehen Kommentare
% Ueberschrift in die Mitte:
\centerline{\bf Freude an \LaTeX}

Normaler Text wird ohne Besonderheiten und ohne auf den rechten Rand
zu achten eingegeben, wenn man mit den Standardeinstellungen zufrieden
ist. Mit Umlauten
"a, "o, "u und mit "s mu"s man sich etwas mehr M"uhe geben. Il n'est
pas difficile
d'\'ecrire un texte fran\c cais. M\^eme le mot "`l'\oe uf"' ne pose aucun
probl\`eme.

Ein neuer Absatz erleichtert die Lesbarkeit. Mathematische Formeln sind ganz
logisch aufgebaut und leicht zu lernen:
$$ \int_0^\infty e^{-x^2}dx={\frac{\sqrt\pi}{2}} $$
Matrizen werden einfach elementweise mit dem entsprechenden Kommando
eingegeben:
$${\bf T}_v^{-1}=\pmatrix{q&-qt_{12}&0\cr % Trennung der Elemente durch &
-qt_{21}&q&0\cr % Ende der Matrixzeile durch \cr
0&0&1}$$ % Eingabe ohne Ruecksicht auf Schoenheit
und dann automatisch in optisch zufriedenstellender Weise ausgedruckt !!
\end{document}
```

erzeugt folgende Ausgabe

Freude an LATEX

[5] Für dieses Kapitel wurden bei Benutzung des Makropakets LATEX ca. 81 unterschiedliche Befehle benutzt.

Normaler Text wird ohne Besonderheiten und ohne auf den rechten Rand zu achten eingegeben, wenn man mit den Standardeinstellungen zufrieden ist. Mit Umlauten ä, ö, ü und mit ß muß man sich etwas mehr Mühe geben. Il n'est pas difficile d'écrire un texte français. Même le mot „l'œuf“ ne pose aucun problème.

Ein neuer Absatz erleichtert die Lesbarkeit. Mathematische Formeln sind ganz logisch aufgebaut und leicht zu lernen:

$$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}$$

Matrizen werden einfach elementweise mit dem entsprechenden Kommando eingegeben:

$$\mathbf{T}_v^{-1} = \begin{pmatrix} q & -qt_{12} & 0 \\ -qt_{21} & q & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

und dann automatisch in optisch zufriedenstellender Weise ausgedruckt !!

11.4.2 Erstellen eines LaTeX–Dokumentes

1. Beim Erstellen eines LaTeX–Dokumentes unter Linux wird dieses zuerst in LaTeX–Syntax in einer Datei mit der Endung *.tex verfaßt.

2. Die Datei wird mit dem Befehl

   ```
   latex datei.tex
   ```

   in eine DVI–Datei mit entsprechender Endung konvertiert. Dieses muß evtl. zweimal geschehen, da z.B. das Inhaltsverzeichnis erst beim zweiten Übersetzen korrekt umgesetzt wird.

3. Das vorläufige Ergebnis kann mit dem Befehl

   ```
   xdvi datei.dvi &
   ```

   angesehen werden. XDVI aktualisiert das Dokument automatisch, wenn die LaTeX–Datei erneut übersetzt wird.

4. Um aus einer DVI–Datei eine Postscript–Datei zu erzeugen, verwendet man

   ```
   dvips datei.dvi
   ```

   Dadurch erhält man eine *.ps–Datei, die wie gewohnt mit dem Befehl `gv datei.ps &` auf dem Bildschirm dargestellt bzw. mit `lpr datei.ps` gedruckt werden kann.

Zur komfortableren Bearbeitung und Übersetzung von LaTeX–Dokumenten können unter Linux auch die Tools `ttem` oder, als graphische Version, `xtem` genutzt werden.

11.5 Zusammenfassung

Textverarbeitung auf „intelligenten“ Schreibmaschinen hat den Nachteil, daß diese Geräte ausschließlich für diesen Zweck benutzt werden können. Will man auf dem Computer arbeiten, muß man sich

zwischen Textverarbeitungsprogrammen, Textformatierern und Desktop–Publishing DTP entscheiden. DTP–Systeme sind im allgemeinen zu teuer und kompliziert. Bei Textverarbeitungsprogrammen können mangelnde Portabilität und Formelgenerierung zum Problem werden. Attraktiv ist, daß Texterstellung, -formatierung und -ausdruck in eine Benutzeroberfläche integriert sind. Bei den Textformatierern hat sich LaTeX durchgesetzt, welches sich durch Portabilität, elegante Formelgenerierung und bestmögliches Schriftbild auszeichnet. Zudem ist LaTeX Public Domain.

11.6 Fragen und Übungen zu Kapitel 11

F 11.1 Welche Textverarbeitung ist für Sie am besten geeignet?

F 11.2 Was ist der Unterschied zwischen einem Textformatierer und einem Textverarbeitungsprogramm?

F 11.3 Welche Vorteile hat LaTeX gegenüber Textverarbeitungsprogrammen?

F 11.4 Welche Vor- und Nachteile hat DTP?

F 11.5 Welche Punkte sind bei Auswahl einer Textverarbeitung zu beachten?

11.7 Antworten zu den Fragen und Übungen zu Kapitel 11

A 11.1 Eine allgemeingültige Antwort gibt es nicht. Man muß viele Faktoren abwägen.

A 11.2 Ein Textverarbeitungsprogramm umfaßt die Funktionen Texterstellung, -formatierung und -ausdruck in einer Benutzeroberfläche. Textformatierer hingegen bearbeiten eine Datei, wobei offen ist, woher die Datei kommt.

A 11.3 Portabilität, elegante Formelgenerierung, bestmögliches Schriftbild, Public Domain

A 11.4 Vorteile: alles ist möglich

Nachteile: teuer, nicht portabel, kompliziert, alles ist möglich

A 11.5 Siehe Seite 163.

Tabellenverzeichnis

1.1 Entwicklung von Prozessoren der Intel Corp. (Auswahl) 4

3.1 Dienste des Internet 54

5.1 Darstellung von Informationseinheiten 72
5.2 Zeichentabelle des ASCII mit Hexadezimal und Binärcode 73
5.3 Dezimale, duale und hexadezimale Zahlen 74
5.4 Zahlbereiche 75
5.5 Darstellung des Exponenten bei REAL–Zahlen 77
5.6 Der BCD–Code 78

6.1 Kategorien von Rechnern und ihre Anwendungen. 82

7.1 Allgemeine Anschlußschnittstelle eines Mikroprozessors [10] 105
7.2 Übersicht über Mikroprozessor–Familien 111

8.1 Belegung der Pins des 25–pol–Steckers 131
8.2 Belegung der Pins des 9–pol–Steckers 132
8.3 Die Standard–Übertragungsgeschwindigkeiten in Baud 133

9.1 Kategorien von Netzen (nach der Entfernung eingeteilt) 139
9.2 Das OSI–Schichtenmodell 140
9.3 Übertragungsmedien von Rechnernetzwerken 144

Abbildungsverzeichnis

1.1 Das S1672–Motherboard der Firma Tacoma [18] 2

1.2 Prozessor [19] 3

1.3 Größe eines Prozessors [20] 3

1.4 Entwicklungsetappen für Großrechner (Mainframe), Kleinrechner(Minicomputer) und Mikrorechner/ Mikroprozessoren [10] 6

3.1 Schalenmodell eines Betriebssystems 28

3.2 Beispiel eines Dateibaumes 35

3.3 Zeitlicher Verlauf von Prozessen 42

3.4 mögliche Speicherhierarchie eines PCs 44

3.5 Die verschiedenen Schichten zwischen Benutzer und Hardware 45

3.6 Netzanbindung des Rechnerpools 53

3.7 Überführung des Quellcodes in ein ausführbares Programm 65

5.1 Abspeicherung einer REAL–Zahl nach Mikroprozessornorm 77

5.2 Abspeicherung einer doppelt genauen Zahl 77

6.1 PC — Personal Computer 82

6.2 schematischer Aufbau eines Motherboards 84

6.3 vereinfachter schematischer Aufbau eines Prozessors 86

6.4 Diskettenlaufwerk 89

6.5 Darstellung einer Festplatte 90

6.6 Festplatte 90

6.7 Grafikkarte 91

6.8 Zeichendarstellung in einer Punktmatrix 92

6.9 Tintenstrahldrucker 93

6.10 Laserdrucker 93

6.11 Modem 94

6.12 ISDN–Karte 94

6.13 CD–ROM 96

7.1 Blockschaltbild des Intel 8080 103

7.2 Befehlsablauf in einem Von–Neumann–Rechner für Mehrwortbefehle ohne Beachtung von Adreßrechnungen und Ein-/Ausgabeoperationen. 106

7.3 Befehlszyklus für einen Speicherzugriff (Lesen) [10] 107

7.4 Darstellung von Schiebe- und Rotationsbefehlen 112

7.5 Mikroprozessor 8080; a: Anschlüsse; b: Registerstruktur 114

8.1 Schematische Darstellung einer Rechner–Drucker–Verbindung. 126

8.2 Leitungsgruppen der Centronics–Schnittstelle auf der Anschlußseite des 36–poligen Steckers und des 25–poligen Subminiatur–Steckers. 129

8.3 Centronics–Timing mit BUSY nach c't 1/84. 130

8.4 Beispiel für eine serielle asynchrone Übertragung der ASCII–Ziffer 9 (39_{16}) mit Startbit, einem Stopbit und gerader Parität. (nach c't 12/86) 131

8.5 Normstecker 25–polig(rechts), 9–pol–Ausführung (links). Mindestausstattung für bidirektionalen Transfer (Dreileitungsverbindung). 131

9.1 Wichtige LAN–Topologien 141

9.2 Das OSI–Schichtenmodell 143

9.3 Das OSI–Referenzmodell im Vergleich zum Internet–Schichtenmodell 146

9.4 Beispielhafte Dienste des Internet–Schichtenmodells 146

10.1 Struktur eines typischen Grafiksystems 152

10.2 historische Entwicklung von Grafikstandards 153

10.3 Struktur eines Grafikssystems mit OpenGL 153

10.4 Output des PostScript–Programm 157

Literaturverzeichnis

[1] *Byte*. McGraw–Hill, New York, monatlich erscheinende Computerzeitschrift

[2] *c't* — Computerfachzeitschrift, Heise–Verlag, Hannover

[3] *Duden 'Informatik'*. Dudenverlag Mannheim/Wien/Zürich, 1988

[4] *Buchdruck auf dem PC*, S. Demmig, Juli 1989

[5] *Textverarbeitung von Rang — acht prominente Programme*, A. Fourier, S. Pfeiffer, Mai 1990

[6] *Informatik für Ingenieure*, Prof. Dr. Dr. E. Hering, Dipl.–Phys. Dr. U. Dyllong, Dipl.–Ing. J. Gutekunst, VDI–Verlag GmbH, Düsseldorf 1995

[7] *The TEXbook*, D. E. Knuth, Addison–Wesley, 1986

[8] *LATEX— Eine Einführung*, H. Kopka, Addison–Wesley, 1994

[9] *LATEX— A Document Preparation System*, L. Lamport, Addison–Wesley, 1988

[10] *Mikroprozessoren und Mikrorechner*, W. Meiling und Ruprecht Fülle, Akademie–Verlag Berlin, 1988

[11] *Fortran95 Sprachumfang*, Regionales Rechenzentrun für Niedersachsen (RRZN), 2. Auflage, RRZN Hannover, 1997 (erhältlich im Sekretariat der ZRZ, TUB)

[12] *Beiträge zur Rechnerarithmetik* RRZN-Bericht 38, 2. rev. Auflage, Regionales Rechenzentrum für Niedersachsen, 1988.

[13] *Computer Graphics: System and Concepts*, R. Salmon und M. Slater, Addison–Wesley, Massachusetts, 1987

[14] *Einführung in TEX*, Schwarz, Addison–Wesley, 1988

[15] *Computer Networks*, A.S. Tanenbaum, Prentice–Hall, 1996

[16] *VDI–Nachrichten*,Wochenzeitung des VDI, VDI–Verlag, Düsseldorf

[17] *PostScript Language*, Reference Manual (1986); Language Tutorial and Cook Book (1986); Language Design (1988); alle Bände: Addison–Wesley, Reading, Massachusetts.

[18] *http://www.mssi.com/s1672pic.htm*

[19] *http://www.intel.com*

[20] *http://www.intel.com/pressroom/archive/releases/dp010897.htm*